

UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR



Doble grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Predicción de elecciones políticas mediante aprendizaje automático

Pablo Ajo Inglez
Tutor: David Renato Domínguez Carreta

Julio 2020

Predicción de elecciones políticas mediante aprendizaje automático

1819_1619_COISSE

AUTOR: Pablo Ajo Inglez

TUTOR: David Renato Domínguez Carreta

Dpto. Ingeniería Informática

Doble grado de Ingeniería Informática y Matemáticas

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2020

Resumen (castellano)

En este trabajo se aplican técnicas de Aprendizaje automático con el objetivo de realizar predicciones en elecciones políticas. Las elecciones políticas sobre las que se trabaja son las Elecciones al Presidente de gobierno en España desde el año 2000. También se consideran, aunque con menor detalle, las últimas Elecciones al Presidente de gobierno en Brasil y Estados Unidos. El objetivo del trabajo es predecir la orientación política (izquierda, derecha, regionalista...) de una población a partir de un conjunto de características de esta, como la densidad de población, la religiosidad, el porcentaje de mujeres... Para realizar estas predicciones se han considerado problemas de regresión y problemas de clasificación. Los métodos de Aprendizaje automático utilizados son regresiones lineales mediante línea y parábola, el Algoritmo KNN, Redes neuronales Single-layer y Redes neuronales Multi-layer. Para cada uno de estos métodos se han realizado distintas ejecuciones variando sus parámetros e hiperparámetros. Los algoritmos se han implementado utilizando el lenguaje de programación Python y se han ejecutado tanto el local como en la plataforma Google Colab. Las librerías más utilizadas en la implementación han sido Keras, Pandas, Sklearn, NumPy y Tensorflow. Se han obtenido métricas para cada ejecución de los algoritmos. El rendimiento de los algoritmos para cada problema considerado es mostrado y comparado en el trabajo.

Abstract (English)

In this work, Machine Learning techniques are applied with the goal of making predictions in electoral processes. The main electoral processes that we consider are the Elections for the President of the Government in Spain since 2000. The last Elections for the President of the Government in Brazil and the United States are also considered, although in less detail. The main goal of the work is to predict the political orientation (left, right, regionalist ...) of a population based on a set of attributes such as population density, religiosity, the percentage of women ... To make these predictions regression problems and classification problems have been considered. The Machine learning methods used are linear regressions using line and parabola, the KNN algorithm, Single-layer neural networks and Multi-layer neural networks. For each of these methods, different executions have been carried out, varying their parameters and hyperparameters. The algorithms have been implemented using the Python programming language and have been executed both locally and on the Google Colab platform. The most used libraries in the implementation have been Keras, Pandas, Sklearn, NumPy and Tensorflow. Metrics have been obtained for each execution of the algorithms. The performance of the algorithms for each problem considered is shown and compared in the work.

Palabras clave (castellano)

Aprendizaje, automático, elecciones, política, red, neuronal, knn, regresión, lineal, parábola, Python...

Keywords (inglés)

Machine, learning, elections, political, network, neural, knn, regression, linear, parabola, Python...

Agradecimientos

Quisiera agradecer a mi madre, mi padre y mi hermano el haberme apoyado y dado cariño durante toda mi vida y en especial durante la realización del Doble Grado. También les agradezco haber despertado en mí el interés por las ciencias y la tecnología.

Agradecer a Carina Barcella por facilitarme la fuente de donde obtener los datos relativos a las elecciones políticas brasileñas.

INDICE DE CONTENIDOS

1	Introducción.....	1
1.1	Motivación.....	1
1.2	Objetivos.....	1
1.3	Organización de la memoria.....	2
2	Estado del arte	5
2.1	Elecciones políticas	5
2.2	Aprendizaje automático.....	5
2.2.1	Historia	5
2.2.2	Regresión lineal	6
2.2.3	Algoritmo K-NN	7
2.2.4	Redes neuronales artificiales	8
2.2.4.1	Redes neuronales artificiales	8
2.2.4.2	Stochastic gradient descent (SGD)	10
2.2.5	Medidas	11
3	Diseño.....	13
3.1	Bases de datos.....	13
3.1.1	Elecciones al presidente de gobierno en España	13
3.1.2	Elecciones al presidente de gobierno en Brasil, Estados Unidos y España..	15
3.2	Modelos de Aprendizaje automático	17
3.2.1	Problemas considerados	17
3.2.2	Regresión lineal	18
3.2.3	Algoritmo K-NN	19
3.2.4	Red neuronal Single-layer	19
3.2.5	Red neuronal Multi-Layer	20
3.3	Flujo de trabajo	21
4	Desarrollo	23
4.1	Recopilación de los datos	23
4.2	Construcción de la base de datos	24
4.2.1	Preprocesamiento de los datos.....	24
4.2.2	Construcción de la base de datos PostgreSQL	24
4.3	Implementación del código	25
4.3.1	Python.....	25
4.3.2	Google Colab.....	25
4.3.3	Pandas, Keras, Sklearn y Tensorflow	26
5	Integración, pruebas y resultados	27
5.1	Estudio estadístico básico	27
5.2	Regresión lineal	28
5.2.1	Regresión lineal mediante línea.....	28
5.2.2	Regresión lineal mediante parábola.....	30
5.3	Algoritmo K-NN	31
5.4	Red neuronal Single-Layer.....	33
5.5	Red neuronal Multi-Layer	35
5.6	Comparación entre los modelos utilizados para los problemas de clasificación....	37
6	Conclusiones y trabajo futuro.....	38
6.1	Conclusiones.....	38
6.2	Trabajo futuro	38
	Referencias	39

Anexos.....	XLIII
A	Normas de estilo utilizadas para la introducción de código en la memoriaXLIII
B	Tablas de los cinco partidos principales de Brasil, Estados Unidos y EspañaXLIII
C	Identificadores numéricos característicasL
D	Script Python para la obtención del número de habitantes de cada municipioLI
E	Clasificación de los partidos políticos LII
F	Función carga_datos_csv.....LVI
G	Matrices de covarianzas, varianzas, medias, máximos y mínimosLIX
H	Gráficos de las regresiones lineales.....LXIX
I	Tablas resultados Algoritmo KNN.....LXXXVI
J	Resultados Red neuronal Single-layer..... XCI
K	Resultados Red neuronal Multi-layerXCVI
L	Código Python CIV

INDICE DE FIGURAS

FIGURA 2-1 EJEMPLO DE REGRESIÓN LINEAL SIMPLE	7
FIGURA 2-2 NEURONA.....	8
FIGURA 2-3 RED NEURONAL	9
FIGURA 2-4 EJEMPLO DE RED NEURONAL	9
FIGURA 2-5 EJEMPLO DE MATRIZ DE CONFUSIÓN	11
FIGURA 3-1 ALGUNOS REGISTROS DE LA BASE DE DATOS PRINCIPAL.....	15
FIGURA 3-2 ALGUNOS REGISTROS DE LA BASE DE DATOS SECUNDARIA	16
FIGURA 3-3 EJEMPLO DE REGRESIÓN LINEAL MEDIANTE PARÁBOLA	19
FIGURA 3-4 RED NEURONAL DE UNA CAPA DEL PROBLEMA DE CLASIFICACIÓN 3	20
FIGURA 3-5 RED NEURONAL MULTICAPA DEL PROBLEMA DE CLASIFICACIÓN 3	21
FIGURA 5-1 RESULTADOS KNN PROBLEMA DE CLASIFICACIÓN 1	32
FIGURA 5-2 RESULTADOS KNN PROBLEMA DE CLASIFICACIÓN 2	32
FIGURA 5-3 RESULTADOS KNN PROBLEMA DE CLASIFICACIÓN 3	33
FIGURA 5-4 OVERFITTING	34
FIGURA 5-5 ACCURACY POR TASA DE APRENDIZAJE, RED NEURONAL MULTI-LAYER.....	36

INDICE DE TABLAS

TABLA 4-1 FUENTES BASE DE DATOS.....	23
TABLA 5-1 MEDIA, VARIANZA, MÁXIMOS Y MÍNIMOS ELECCIONES GOBIERNO DE ESPAÑA	27
TABLA 5-2 VARIABLES CON MÁS CORRELACIÓN PROBLEMA DE REGRESIÓN 1	27
TABLA 5-3 RESULTADOS REGRESIÓN LINEAL MEDIANTE LÍNEA PROBLEMA DE REGRESIÓN 4.....	28
TABLA 5-4 RESULTADOS REGRESIÓN LINEAL MEDIANTE LÍNEA PROBLEMA DE REGRESIÓN 1.....	29
TABLA 5-5 RESULTADOS REGRESIÓN LINEAL MEDIANTE LÍNEA PROBLEMA DE REGRESIÓN 2.....	29
TABLA 5-6 RESULTADOS REGRESIÓN LINEAL MEDIANTE LÍNEA PROBLEMA DE REGRESIÓN 3.....	29
TABLA 5-7 RESULTADOS REGRESIÓN LINEAL MEDIANTE PARÁBOLA PROBLEMA DE REGRESIÓN 4	30
TABLA 5-8 RESULTADOS REGRESIÓN LINEAL MEDIANTE PARÁBOLA PROBLEMA DE REGRESIÓN 1	30
TABLA 5-9 RESULTADOS REGRESIÓN LINEAL MEDIANTE PARÁBOLA PROBLEMA DE REGRESIÓN 2	31
TABLA 5-10 RESULTADOS REGRESIÓN LINEAL MEDIANTE PARÁBOLA PROBLEMA DE REGRESIÓN 3	31

1 Introducción

1.1 Motivación

En la política, en concreto, en una democracia representativa, las elecciones electorales son el proceso mediante el cual los electores (ciudadanos) escogen mediante su voto entre un conjunto de candidatos cuáles de estos ocuparán unos determinados cargos políticos. Existen diferentes sistemas electorales (conjunto de reglas que determinan cómo se lleva a cabo la elección) y existen elecciones a diferentes cargos políticos. En este trabajo nos centraremos en las elecciones a presidentes de gobierno (jefes de estado). La mayoría de los datos tratados harán referencia a las elecciones a presidente de gobierno en España, aunque también se consideran datos de Brasil y Estados Unidos.

El marco político español actual está marcado por el fin del bipartidismo aproximadamente desde 2014, con la aparición en escena de nuevos partidos políticos de diferentes ideologías. Otro factor importante para comprender la situación política española actual es la crisis del nacionalismo catalán, que alcanzó su momento de máxima tensión en el referéndum de independencia celebrado en 2017.

La inteligencia artificial juega un papel cada vez más importante en la democracia. Los partidos políticos hacen uso de diferentes algoritmos para dirigir mensajes a sus potenciales electores, tal y como lo haría una empresa privada con sus potenciales consumidores. Los partidos políticos españoles apuestan cada vez más por la propaganda digital, lo cual quiere decir que invierten parte de su dinero en publicidad dirigida (mediante inteligencia artificial). En este trabajo se hace uso de diferentes métodos de aprendizaje automático para intentar predecir el partido más votado por una población a partir de ciertas características. Herramientas como esta jugarán un papel fundamental en las democracias del futuro.

1.2 Objetivos

El objetivo de este trabajo es realizar predicciones sobre elecciones políticas utilizando distintos algoritmos de aprendizaje automático en función de distintas características de una población, tales como la edad media, el PIB per cápita... El objetivo es predecir el partido más votado en dichas poblaciones. Debido a que en diferentes comunidades, unidades federales y países existen diferentes partidos, estos han sido clasificados en función de su posición en el espectro político (izquierda-derecha) para poder manejar clases comunes en distintas poblaciones.

Los datos sobre los que hemos trabajado se pueden dividir en dos bloques. En primer lugar, la base de datos principal, que se centra en las elecciones a presidente del gobierno en España. La división territorial mínima que se ha considerado ha sido la mesa electoral, es decir, cada “fila” de la base de datos representa una mesa electoral en un proceso electoral. Sobre esta base de datos hemos considerado hasta 22 características de la población tales como la edad media, el porcentaje de mujeres, la religiosidad... Además de este conjunto de datos, con el objetivo de incluir también otros países, hemos construido otra base de datos que recoge información sobre las elecciones a presidente de gobierno en Brasil, Estados Unidos y España. Esta base de datos es de menor tamaño, las divisiones territoriales son los

estados o provincias y la única característica considerada ha sido el IDH (índice de desarrollo humano) de las poblaciones. Sobre el primer conjunto de datos, el principal, se han considerado tres clasificaciones diferentes. En la primera de ellas clasificamos los partidos en los siguientes grupos: Izquierda, Centro-izquierda, Centro-derecha, Derecha, Regionalista-izquierda, Regionalista-derecha. En la segunda sólo distinguimos entre Izquierda y Derecha. En la tercera, los partidos se clasifican entre las clases Regionalista y No regionalista.

Sobre el conjunto de datos principal mencionado anteriormente hemos usado los siguientes algoritmos. En primer lugar, predicciones utilizando una regresión lineal, usando una línea y tras esto una parábola. En segundo lugar, el algoritmo de vecino próximos K-NN. Por último, y como materia principal del trabajo, hemos considerado una red neuronal sin capas intermedias y después una red neuronal con una capa intermedia. Hemos realizado comparaciones internas de los distintos algoritmos variando sus parámetros. Tras esto, hemos realizado una comparación entre los distintos algoritmos utilizados.

Para la implementación de los algoritmos se ha utilizado el lenguaje Python, en su versión 3. Los programas han sido ejecutados tanto en local en el sistema operativo Ubuntu como en la plataforma Google Colab. Se destaca el uso de las librerías NumPy, Pandas, Keras y Tensorflow. Los datos han sido gestionados y almacenados en una base de datos PostgreSQL y el gestor de bases de datos utilizado pgAdmin.

Para finalizar, en el último capítulo de esta memoria sintetizamos el trabajo realizado y reflexionamos sobre el posible trabajo a realizar después de este TFG.

1.3 Organización de la memoria

La memoria está formada por los siguientes capítulos:

- **Introducción.** Capítulo introductorio al trabajo realizado. En él mencionamos los objetivos del trabajo, la motivación de la realización de este y la organización de la memoria.
- **Estado del arte.** En este capítulo realizamos una introducción teórica a los algoritmos de aprendizaje automático utilizados en el trabajo, así como un resumen de su historia.
- **Diseño.** En este capítulo presentamos con detalle las dos bases de datos que hemos elaborado y las decisiones de diseño que hemos tomado tanto en su construcción como en la construcción de los algoritmos. También en este capítulo se detalla el flujo de trabajo que se ha seguido en la realización de este TFG.
- **Desarrollo.** En el capítulo de desarrollo detallamos cómo ha sido realizada la recopilación de los datos sobre los que se ha trabajado, así como las fuentes

utilizadas. Detallamos también la implementación de la base de datos y del código. Hablamos aquí también sobre las librerías utilizadas.

- **Integración, pruebas y resultados.** Este capítulo es el principal del trabajo. En él se muestran los resultados obtenidos para los diferentes algoritmos en función de los parámetros con los que han sido ejecutados. Finalmente se realiza una comparación entre los diferentes algoritmos, valorando distintos escenarios.
- **Conclusiones y trabajo futuro.** Recapitulación del trabajo realizado y reflexión sobre las posibles mejoras a realizar.

2 Estado del arte

2.1 Elecciones políticas

Como se ha definido en la sección Motivación, en la política, en concreto, en una democracia representativa, las elecciones electorales son el proceso mediante el cual los electores (ciudadanos) escogen mediante su voto entre un conjunto de candidatos cuáles de estos ocuparán determinados cargos políticos. En este trabajo nos centramos en las elecciones a presidente de Gobierno en España, aunque también se tiene en cuenta (aunque con menos importancia) elecciones al presidente de gobierno en Brasil y en Estados Unidos.

Tras la dictadura, en España, se han celebrado catorce elecciones generales desde la promulgación de la Constitución de 1978, la más reciente en 2019. En estas elecciones los ciudadanos escogen a los miembros del Consejo de los Diputados y del Senado, aunque en este trabajo nos centraremos en los resultados que hacen referencia al Consejo de los Diputados. Los miembros de esta cámara se eligen mediante un sistema de representación proporcional. El territorio español está dividido en cincuenta y dos circunscripciones, que son las cincuenta provincias y a las ciudades autónomas de Ceuta y Melilla. El sistema utilizado para repartir los escaños es el sistema D'Hont, cuyo funcionamiento no procede detallar. En cada proceso electoral los ciudadanos acuden a su mesa electoral para emitir su voto. Cada mesa electoral comprende entre quinientos y dos mil ciudadanos. Esta, la mesa electoral, es la unidad mínima de población considerada en el trabajo.

Los datos tratados hacen referencia a las elecciones celebradas desde el 2000 (incluido). Desde este año, los acontecimientos que más han marcado la política española a nivel nacional son los siguientes. En primer lugar, la crisis financiera de 2008. En segundo lugar, el llamado “fin del bipartidismo” en 2014. Anteriormente, en el escenario político español dos partidos destacaban sobre el resto, el Partido Popular y el Partido Socialista. En 2014, un nuevo partido político, Podemos, supera en intención de voto al Partido Socialista. Desde entonces otros partidos como Ciudadanos o Vox han irrumpido con fuerza, siendo ya no tan clara la supremacía de PP y PSOE respecto al resto de partidos. Por último, otro suceso importante ha sido la crisis de la independencia de Cataluña, que tuvo su momento de máxima tensión en el referéndum de independencia celebrado en 2017. Es por esto que parte del trabajo está enfocado en predecir la intención de voto a partidos regionalistas frente a no regionalistas (o viceversa).

2.2 Aprendizaje automático

2.2.1 Historia

El Aprendizaje automático es una rama de la Inteligencia artificial que tiene como objetivo la elaboración de algoritmos, modelos y técnicas que permitan “aprender” a los sistemas informáticos.

La historia del aprendizaje automático empieza en 1950 cuando Alan Turing publica su famoso artículo “Computing machinery and intelligence” en el que plantea la Prueba de Turing. En 1952 Arthur Samuel escribe el primer programa capaz de aprender, un software

que jugaba a las damas y aprendía de sus errores. En 1979 estudiantes de la Universidad de Stanford son capaces de diseñar un vehículo que es capaz de moverse autónomamente por una habitación evitando obstáculos. En 1981 Gerald Dejong crea el término Aprendizaje basado en experiencias. A principios de la década de los 90 los científicos empiezan a crear programas que analizan grandes cantidades de datos y son capaces de obtener conclusiones, aprendiendo así de los resultados. En 1996 tiene lugar la famosa partida de ajedrez entre Gary Kasparov y Deep Blue, una computadora de IBM, en la que la segunda sale vencedora. En 2006 Geoffrey Hinton presenta el concepto de Deep Learning.

Si nos centramos en el campo de las redes neuronales, que vive dentro del Aprendizaje automático y es el objeto principal de estudio de este trabajo, podemos destacar los siguientes acontecimientos históricos. En 1958 se crea el Perceptron, que es ampliado en 1965 al Perceptron multicapa. En la década de los 80 aparecen las Neuronas sigmoidales, las Redes feedforward y el algoritmo Backpropagation. En 1989 nacen las Redes neuronales recurrentes y las Redes neuronales convolucionales. En 1997 aparecen las LSTM (Long short term memory), que son un caso particular de redes neuronales recurrentes. En 2006, como hemos mencionado en el párrafo anterior, nace el Deep learning con la aparición del Auto-encoder y la máquina de Boltzmann. En 2014 son inventadas las Redes generativas antagónicas.

Hoy en día el Aprendizaje automático, a pesar de ser un campo joven, posee una gran fama e importancia. Tiene una gran cantidad de aplicaciones en la industria tales como reconocimiento de voz e imágenes, construcción de coches autónomos, aplicaciones en medicina, juegos... Además de esto, también es el campo en el que trabajan una gran cantidad de investigadores, desde un enfoque más científico.

2.2.2 Regresión lineal

La regresión lineal tiene como finalidad la construcción de un modelo que permita explicar la relación que existe entre un conjunto de variables independientes y una variable dependiente. Este modelo es descrito mediante una ecuación que es lineal en sus coeficientes. Debido a esto, mediante una regresión lineal, la relación entre las dos variables anteriormente mencionadas puede ser explicada mediante una recta, una parábola o un polinomio de grado mayor. Explicaremos en detalle el caso en el cual sólo existe una variable independiente. Si existieran más variables independientes los pasos a seguir siguen el mismo enfoque. El modelo de regresión lineal simple puede ser descrito mediante la siguiente ecuación.

$$Y = a_0 + a_1X_1 + \varepsilon$$

En esta ecuación Y es la variable dependiente, X_1 es la variable independiente, a_0 es la ordenada en el origen, a_1 es la pendiente y ε el error, que representa la diferencia entre el ajuste (por donde pasa la recta) y el valor real. El método de regresión lineal consiste en hallar los estimadores \hat{a}_0 y \hat{a}_1 que minimicen el error. Tomaremos como error a la suma de los cuadrados residuales, es decir, la suma de los cuadrados de las diferencias entre los valores reales y los valores estimados. Esto da lugar a la recta que mejor aproxima la nube de puntos. Esta recta viene dada por la ecuación

$$\hat{y} = \hat{a}_0 + \hat{a}_1x$$

Los coeficientes \hat{a}_0 y \hat{a}_1 son obtenidos, por ejemplo, derivando el error en función de \hat{a}_0 y \hat{a}_1 y hallando los mínimos. El valor de estos resulta ser:

$$\hat{a}_1 = \frac{\sum_{i=1}^n (x_i - E[x])(y_i - E[y])}{\sum_{i=1}^n (x_i - E[x])^2}$$

$$\hat{a}_0 = E[y] - \hat{a}_1 E[x]$$

A continuación mostramos un ejemplo de una regresión lineal realizada durante el trabajo consistente en la predicción del porcentaje de votos a la izquierda en España tomando como variable independiente la religiosidad (cuya descripción detallada veremos más adelante). Los puntos azules representan los valores reales y la recta roja la predicción realizada mediante la regresión lineal simple.

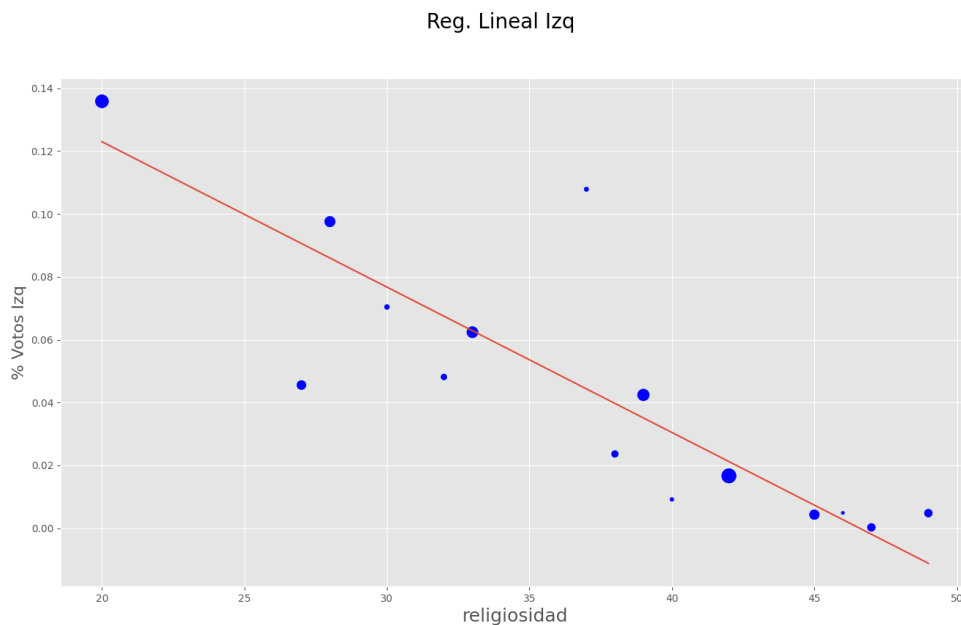


Figura 2-1 Ejemplo de regresión lineal simple

2.2.3 Algoritmo K-NN

El *Algoritmo K-NN* o *Método de los K vecinos más cercanos*, es un método de clasificación supervisada. A continuación explicamos su funcionamiento. El algoritmo requiere de conjunto de entrenamiento. El conjunto de entrenamiento está formado por vectores en un espacio multidimensional. Cada ejemplo del conjunto de entrenamiento está formado por p atributos y pertenece a una clase. Pueden existir dos o más clases. Es decir, cada ejemplo es un vector $(x_1, \dots, x_p) \in \mathbb{R}^p$ que pertenece a una de q clases. Detallaremos a continuación las fases del algoritmo.

Fase de entrenamiento. La fase de entrenamiento es sencilla y consiste en almacenar los vectores de los ejemplos y etiquetarlos con la clase correspondiente.

Fase de clasificación. En esta fase se intenta predecir la clase asociada a los vectores que queremos clasificar (de los que se desconoce la clase). Para realizar esta predicción se calcula la distancia que hay entre los vectores acumulados en la fase de entrenamiento y el vector que se desea clasificar. La clase más repetida entre los K vectores más cercanos es la clase que se le asigna al nuevo vector.

El número K de vecinos que se consideran puede variar. Otro parámetro que puede variar en este algoritmo es la distancia utilizada. Normalmente se utiliza la distancia euclídea.

$$d(x, y) = \sqrt{\sum_{i=1}^p (x_i - y_i)^2}$$

2.2.4 Redes neuronales artificiales

2.2.4.1 Redes neuronales artificiales

Las *redes neuronales artificiales* son un modelo computacional que en un inicio se desarrolló inspirándose en el comportamiento de las neuronas humanas y en la manera de aprender de estas. Este modelo consta de una serie de elementos conectados entre sí llamados *neuronas artificiales*. La conexión entre las diferentes neuronas funciona como un sistema que toma una entrada de tamaño m y entrega una salida de tamaño n, este sistema recibe el nombre de *red neuronal*.

Cada neurona posee una entrada, una salida y una función de activación. Explicaremos que es cada uno de estos elementos. La entrada de una neurona es un vector de valores numéricos, multiplicados cada uno por un peso. La *función de activación* es una función que toma como entrada el sumatorio de los valores del vector multiplicado cada uno por su peso y entrega una salida (normalmente un valor numérico entre 0 y 1 o -1 y 1). Algunas de las funciones de activación más utilizadas son la *Sigmoide*, la *tangente hiperbólica* o la función *Softmax* cuando se trata de un problema de clasificación.

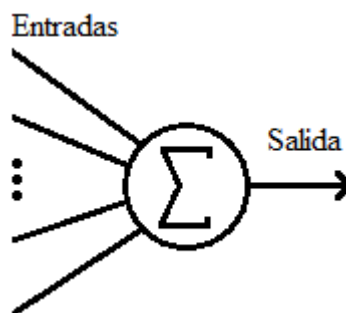


Figura 2-2 Neurona

Cada neurona está conectada con las otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un peso y constituye una de las entradas de la siguiente neurona. Las neuronas se organizan en capas, existiendo una capa de entrada, capas intermedias y una capa de salida.

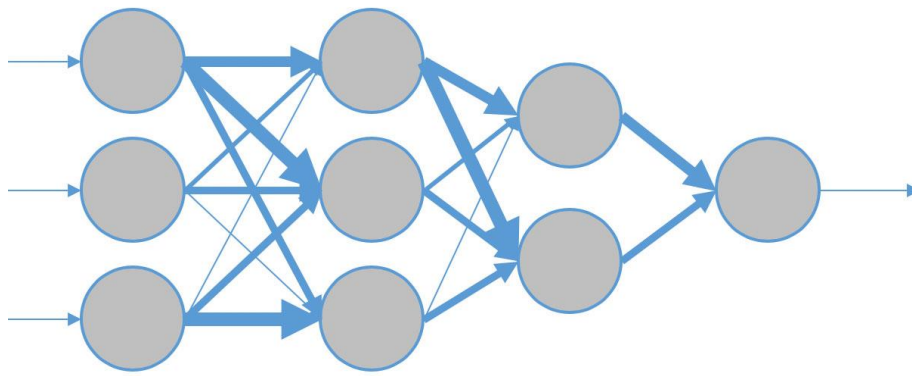


Figura 2-3 Red neuronal

Una red neuronal es capaz de aprender. Este aprendizaje se realiza durante la llamada *fase de entrenamiento* de la red neuronal. Detallaremos el proceso. Partimos de un conjunto de vectores numéricos $(x_1, \dots, x_p) \in \mathbb{R}^p$ los cuales tienen asociados cada uno un vector $(x_1, \dots, x_r) \in \mathbb{R}^r$, unidimensional o multidimensional, que puede representar diversas características codificadas numéricamente, la probabilidad de pertenecer a una entre r clases... Nuestro objetivo es dado un vector $(x_1, \dots, x_p) \in \mathbb{R}^p$, predecir el vector $(x_1, \dots, x_r) \in \mathbb{R}^r$ que tiene asignado. Para ello, en primer lugar, se toma un conjunto de entrenamiento, que es un conjunto de vectores p -dimensionales de los cuales sabemos qué vectores r -dimensionales tienen asignados. Para ilustrar la explicación consideraremos una red neuronal que tiene p neuronas en la capa de entrada, m neuronas en la capa intermedia y r neuronas en la capa de salida. Las neuronas de la capa de entrada no poseen función de activación, es decir, su salida es la misma que la entrada. Cada enlace entre una neurona i de una capa y una neurona j de la siguiente tiene asignado un peso $X_{i,j}$ que funciona como multiplicador de las salidas de las neuronas. Estos pesos son inicializados con ciertos valores aleatorios (normalmente valores pequeños cercanos a 0).

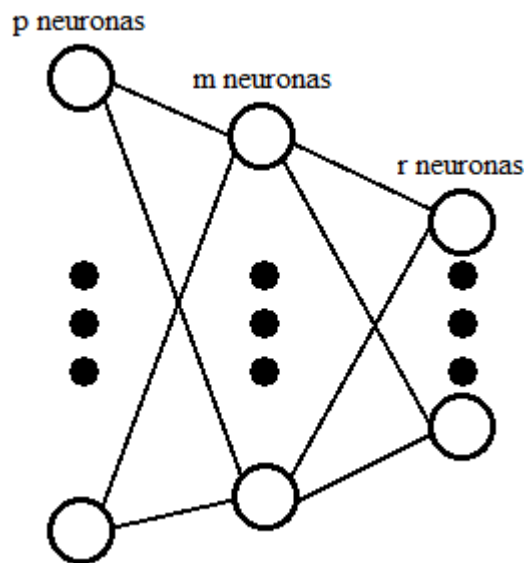


Figura 2-4 Ejemplo de red neuronal

Cada elemento del primer vector del conjunto de entrenamiento es la entrada (y por lo tanto la salida) de una neurona de la primera capa. La segunda capa de neuronas toma como entrada de su función de activación el sumatorio de los valores que recibe multiplicados por los pesos de los enlaces correspondientes y emite un valor en función de esta entrada y su función de activación. Este valor, multiplicado por el peso correspondiente de la conexión con la siguiente capa sirve como entrada para la última capa de neuronas. Estas últimas neuronas producen una salida (la salida correspondiente a su función de activación) que representa el vector $(x_1, \dots, x_r) \in \mathbb{R}^r$, la predicción de la red neuronal para esta muestra (vector de entrada). Este vector salida se toma como entrada de una *función de pérdida*, que evalúa el error del vector predicho respecto del vector real. Los pesos de las conexiones se actualizan tratando de minimizar la suma de estos errores (optimización de la función de pérdida). Este proceso de actualización de los pesos se denomina *Backpropagation*. Existen diversos métodos de optimización de la función de pérdida. Describiremos dos de ellos en la siguiente sección. Se repite este proceso para todos los vectores del conjunto de entrenamiento. El número de vectores que se tiene en cuenta antes de cada actualización de los pesos recibe el nombre de *tamaño del lote*. El número de veces que el conjunto de entrenamiento pasa por la red neuronal en la fase de entrenamiento recibe el nombre de *número de épocas*.

2.2.4.2 Stochastic gradient descent (SGD)

El algoritmo *Stochastic gradient descent (SGD)* es un método de optimización de funciones de pérdida que tiene la propiedad de ser diferenciable. Se trata de una aproximación estocástica al algoritmo de Descenso por gradiente (Gradient Descent), que explicaremos a continuación. La idea de este algoritmo tiene su origen en el Algoritmo de Robbins–Monro, en 1950. Intuitivamente representa la idea de llegar al mínimo de una función descendiendo por la dirección del gradiente (la de máximo decrecimiento).

El algoritmo de Descenso por gradiente tiene como objetivo minimizar una función de la forma $Q(x) = \frac{1}{n} \sum_{i=1}^n Q_i(x)$. Se desea estimar el parámetro x que minimiza $Q(x)$. Cada $Q_i(x)$ en nuestro caso es el valor de la función de pérdida en una observación de la fase de entrenamiento de nuestro algoritmo. En cada iteración del algoritmo de Descenso por Gradiente el valor de x es actualizado del siguiente modo:

$$x_{i+1} = x_i - \eta \nabla Q(x_i)$$

El valor η recibe el nombre de *tasa de aprendizaje* y representa el peso que tiene cada muestra en la actualización del parámetro estimado. Evaluar el gradiente requiere evaluar el gradiente en cada observación Q_i , lo cual puede ser computacionalmente muy costoso. El algoritmo SGD es una variación del algoritmo de Descenso por Gradiente que muestrea un subconjunto de los sumandos en cada iteración, haciendo que el coste computacional se vea reducido.

En la realización del trabajo se ha utilizado el algoritmo *RMSPprop (Root Mean Square Propagation)* que es una ligera variación de SGD en la que la tasa de aprendizaje varía durante la ejecución del algoritmo. Se ha usado RMSPprop variando la tasa de aprendizaje y el *momentum*, que es otro parámetro opcional que actúa como “memoria” de la actualización de x en cada iteración. Además de RMSPprop también se ha usado el algoritmo *Adam*

(*Adaptive Moment Estimation*) que es una variación de RMSProp que hace uso del primer y segundo momento del gradiente en cada iteración, es decir, de la media y la varianza del gradiente.

2.2.5 Medidas

Una *medida* o *métrica* de un modelo de aprendizaje automático es una manera de cuantificar lo bien que el modelo resuelve un problema con determinado objetivo. Existen diferentes problemas y diferentes objetivos, por lo tanto tendremos distintas medidas según nuestras necesidades. Enumeraremos las usadas en este trabajo. En primer lugar distinguiremos entre problemas de regresión y problemas de clasificación. Los problemas de regresión se caracterizan porque intentamos aproximar un valor numérico. En los problemas de clasificación el resultado es una clase entre un número limitado de estas.

Para los problemas de regresión hemos usado como medida la *Raíz del error cuadrático medio normalizado (NRMSE)*. La *Raíz del error cuadrático medio (RMSE)* viene dada por la fórmula

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

donde y_i es cada observación, \hat{y}_i es la estimación. Hay varias maneras de normalizar esta medida. Nos hemos decantado por dividirla por la diferencia entre el valor máximo y mínimo de las observaciones.

En segundo lugar para los problemas de clasificación hemos usado como medidas la *accuracy* y la *precision*, dependiendo del problema. Para explicar en qué consisten estas dos medidas antes debemos explicar lo que es una matriz de confusión. Una *matriz de confusión* es una herramienta que nos permite visualizar cómo un algoritmo ha realizado una clasificación. Cada elemento x_{ij} de la matriz de confusión representa el número de elementos de la clase i que han sido clasificado como elementos de la clase j . Por lo tanto los elementos de la diagonal son los elementos que se han clasificado correctamente.

		Predicted class	
		Cat	Non-cat
Actual class	Cat	5 True Positives	3 False Negatives
	Non-cat	2 False Positives	3 True Negatives

Figura 2-5 Ejemplo de matriz de confusión

A partir de esta matriz se pueden obtener varias medidas. Nos centraremos en las utilizadas en el trabajo. La *accuracy* es la medida más básica y es la proporción de clasificaciones correctas que se han realizado sobre el total. Es una medida que tiene sentido utilizar cuando el número de elementos de cada clase es parecido. En otros casos hemos utilizado la *precision*, que es el porcentaje de clasificaciones que son correctas entre las clasificaciones totales que se han realizado para una clase. Existe una precisión para cada clase. En un problema de clasificación binario en el que por ejemplo, de 100 muestras existen 99 de la

clase A y 1 de la clase B, un algoritmo que clasifique todas como clase A tendrá una *accuracy* del 99%, sin embargo la *precision* de B será del 0%.

3 Diseño

3.1 Bases de datos

3.1.1 Elecciones al presidente de gobierno en España

La base de datos principal del trabajo recopila datos sobre las Elecciones generales de España celebradas desde el 2000 (incluido) en adelante. Los datos han sido obtenidos a través de la web de información electoral del Ministerio de Interior (<http://www.infoelectoral.mir.es/>). Los datos descargados contienen la información de todos los votos de cada proceso electoral divididos por mesa electoral. A partir de estos datos se ha preparado una base de datos PostgreSQL en la que cada fila corresponde a una mesa electoral en un proceso electoral. Las columnas de esta base de datos son las siguientes:

- **Año.** Año de celebración del proceso electoral.
- **Comunidad.** Comunidad a la que pertenece la mesa electoral.
- **Provincia.** Provincia a la que pertenece la mesa electoral.
- **Religiosidad.** Porcentaje de personas que marcaron la casilla de la Iglesia en la comunidad autónoma a la que pertenece la mesa electoral en el año 2016.
- **Edad media.** Edad media de la población en la provincia a la cual pertenece la mesa electoral en el año del proceso electoral.
- **Escolarización.** Porcentaje de población con educación superior en la comunidad autónoma a la que pertenece la mesa electoral.
- **Esperanza de vida.** Esperanza de vida en la provincia a la cual pertenece la mesa electoral.
- **Gobierno de la comunidad.** Posición en el espectro político (Izquierda, Derecha, Centro-izquierda, Centro-derecha, Regionalista-izquierda, Regionalista-derecha) del partido que estaba en el gobierno de la comunidad a la cual pertenece la mesa electoral en el momento en el que ocurrió el proceso electoral correspondiente.
- **Gobierno de España.** Posición en el espectro político (Izquierda, Derecha, Centro-izquierda, Centro-derecha, Regionalista-izquierda, Regionalista-derecha) del partido que estaba en el gobierno de España en el momento en el que ocurrió el proceso electoral correspondiente.
- **Gobierno de Estados Unidos.** Partido político (Demócrata o Republicano) que estaba en el gobierno de Estados Unidos en el momento en el que ocurrió el proceso electoral correspondiente.

- **Habitantes municipio.** Número de habitantes del municipio al cual pertenece la mesa electoral. Sólo un dato temporal, 2011.
- **IDH.** Índice de desarrollo humano en la provincia de la mesa electoral. Sólo un dato temporal, 2007.
- **Acceso a internet.** Porcentaje de cobertura de conexión a internet mayor o igual a 100 mbps en la provincia de la mesa electoral. Sólo un dato temporal, 2018.
- **PIB per cápita.** PIB per cápita en la provincia de la mesa electoral.
- **Criminalidad.** Número de infracciones penales por habitante en la provincia a la cual pertenece la mesa electoral.
- **Inmigración de países en vías de desarrollo.** Porcentaje de habitantes procedentes de África, América Central o América del Sur en la provincia a la que pertenece la mesa electoral.
- **Inmigración europea.** Porcentaje de habitantes procedentes de Europa (sin contar España) en la provincia a la que pertenece la mesa electoral.
- **Viajeros.** Número de viajeros por habitante que visitaron la provincia a la cual pertenece la mesa electoral.
- **Porcentaje de mujeres.** Porcentaje de mujeres sobre el total de habitantes en el municipio al que pertenece la mesa electoral.
- **Latitud.** Latitud geográfica del municipio al que pertenece la mesa electoral.
- **Altitud.** Altitud del municipio al que pertenece la mesa electoral.
- **Longitud.** Longitud geográfica del municipio al que pertenece la mesa electoral.
- **Candidatura.** Esta columna es la más importante, no se trata de una característica, sino que contiene las categorías en las que queremos clasificar cada mesa electoral. El valor original de esta columna era el partido político más votado en la mesa en el proceso electoral correspondiente. Debido a que distintas circunscripciones y distintos años producen diferentes partidos, se han considerado seis categorías en la que se clasifican los partidos políticos. Estas son: Izquierda, Derecha, Centro-izquierda, Centro-derecha, Regionalista-izquierda y Regionalista-derecha. Existen dos copias idénticas de esta base de datos con la única diferencia de que se utilizan categorías diferentes. En la segunda copia se consideran solamente dos categorías, Izquierda y Derecha. En la tercera copia también se consideran solo dos categorías, pero estas son Regionalista y No regionalista.

Si los datos de algún atributo (columna) faltan para algún año, se rellenan con los datos más cercanos temporalmente de los cuales se disponga. La base de datos cuenta con un total de 464085 registros. A continuación adjuntamos una captura de pantalla de algunas filas ordenadas al azar de esta base de datos siguiendo el primer criterio de clasificación de las candidaturas.

	anno integer	comunidad character vary	provincia character va	candidatura character varyi	religiosidad integer	edad_media numeric	escolarizacion numeric	esperanza_vida numeric	gobierno_comunid character varying (2
1	2016	Galicia	Lugo	DER	27	49.36	26.4	82.57	DER
2	2019	Aragón	Zaragoza	DER	38	44.41	30.3	83.44	C-IZQ
3	2008	Baleares	Balears, Il...	C-IZQ	30	39.05	23.0	81.19	C-IZQ
4	2004	Andalucía	Cádiz	C-IZQ	42	36.96	21.3	78.02	C-IZQ
5	2011	Región de ...	Murcia	DER	47	38.41	21.7	81.71	DER
6	2015	Canarias	Palmas, L...	IZQ	28	40.48	21.1	81.73	REGION-DER
7	2016	Cataluña	Barcelona	IZQ	20	42.49	30.0	83.53	REGION-DER
8	2016	País Vasco	Bizkaia	REGION-DER	28	45.42	37.2	83.14	REGION-DER
9	2011	Galicia	Coruña, A	DER	27	44.76	24.5	81.96	DER
10	2000	Galicia	Ourense	DER	27	46.72	24.5	79.90	DER

Figura 3-1 Algunos registros de la base de datos principal

Mediante la instrucción SQL

```
COPY (Select * from TABLE_NAME) To 'ruta/archivo.csv' WITH DELIMITER E
'\t'
```

la base de datos ha sido exportada a un archivo de extensión CSV, que puede ser consumido por las librerías de código utilizadas.

Para el algoritmo de la regresión lineal, al tratarse de un problema de regresión y no de clasificación, se han implementado varias bases de datos adaptadas, más pequeñas, que se ha usado en este caso. En estas nuevas bases de datos cada fila representa un valor distinto de una determinada característica, por ejemplo, los diferentes valores que adquiere la religiosidad. Las columnas por las que están formadas estas tablas son el valor de la característica, el porcentaje de votos para una cierta categoría sobre el total para dicho valor de la característica y el peso (número de registros que tiene ese valor de característica en la base de datos original).

3.1.2 Elecciones al presidente de gobierno en Brasil, Estados Unidos y España

Este segundo conjunto de datos recopila información sobre las últimas elecciones a Presidente de gobierno en Brasil, España y Estados Unidos. Esta tabla contiene menos datos que la descrita en la sección anterior. La finalidad de estos datos es ver, aunque sea de manera mucho menos profunda, el comportamiento de otros países.

En primer lugar se elaboró una tabla por cada país con las siguientes columnas: el nombre de la unidad territorial y el porcentaje de votos a cada uno de los cinco partidos principales del país en dicha unidad territorial. Además cada tabla posee una fila que representa al país entero y contiene las medias de votos a los cinco partidos principales en todo el país. Estas tablas se pueden encontrar en el anexo Tablas de los cinco partidos principales de Brasil,

Estados Unidos y España. También en este anexo se ha calculado la “distancia” de cada unidad territorial a su país.

En segundo lugar se elaboró la tabla principal, que se utiliza en los problemas de regresión. Las columnas de esta segunda tabla son:

- **Nombre unidad territorial.** El nombre de la provincia o estado.
- **Porcentaje de votos a la izquierda.** Porcentaje de votos a la izquierda (considerando solo la suma de los partidos de izquierda entre los cinco partidos principales) sobre el total (todos los partidos) en la unidad territorial. Es uno de los parámetros a predecir mediante regresión.
- **Porcentaje de votos a la derecha.** Porcentaje de votos a la derecha (considerando solo la suma de los partidos de derecha entre los cinco partidos principales) sobre el total (todos los partidos) en la unidad territorial. Es uno de los parámetros a predecir mediante regresión.
- **Índice de desarrollo humano (IDH).** El índice de desarrollo humano de la unidad territorial. Actúa como variable independiente en las regresiones.

En el caso de España cada fila es una provincia. En el caso de Brasil cada fila es un estado. En las filas que hacen referencia a Estados Unidos, cada fila representa un estado. Esta tabla cuenta con 133 registros. Del mismo modo que las tablas de la sección anterior, esta tabla ha sido exportada a un archivo CSV que puede ser consumido por las librerías utilizadas. Adjuntamos una captura de pantalla de algunos registros de este conjunto de datos.

Data Output

	nombre_unidad character varying	izq numeric	der numeric	idh numeric
1	North Dakota	265303752	32344058543	0.942
2	PA	665059170	35370377192	0.698
3	Louisiana	153018779	21356390633	0.885
4	Santa Cruz de Tenerife	605772720	39131679991	0.931
5	North Carolina	209476873	31897154610	0.905
6	Lugo	794014075	32882320235	0.943
7	RO	816777578	34302365652	0.725
8	Toledo	272068911	32823928185	0.933
9	Valencia/València	136212721	77649225059	0.946
10	AP	815826876	54347836476	0.740
11	BRASIL	677821437	38523955923	0.759

Figura 3-2 Algunos registros de la base de datos secundaria

3.2 Modelos de Aprendizaje automático

3.2.1 Problemas considerados

Podemos dividir los algoritmos de aprendizaje automático utilizados en el trabajo en dos grupos, los utilizados para un problema de regresión y los utilizados para un problema de clasificación. La regresión lineal (el algoritmo más simple) es utilizada en problemas de regresión, donde el objetivo es estimar el porcentaje de votos a un determinado rango del espectro político tomando como variable independiente una característica de las almacenadas en la base de datos. La regresión lineal (con línea y parábola) ha sido aplicada sobre los conjuntos de datos descritos en las secciones Elecciones al presidente de gobierno en España y Elecciones al presidente de gobierno en Brasil, Estados Unidos y España.

Uno de los problemas de regresión de este trabajo considera el conjunto de datos Elecciones al presidente de gobierno en Brasil, Estados Unidos y España. En este problema las variables dependientes (las que se intentan predecir) son el porcentaje de votos a izquierda y derecha en una determinada división geográfica considerando los cinco partidos políticos principales de cada país como se ha explicado en la sección Elecciones al presidente de gobierno en Brasil, Estados Unidos y España. En España se consideran las provincias. En Brasil y Estados Unidos se consideran los estados. De ahora en adelante nos referiremos a este problema como *Problema de regresión 4*.

El algoritmo K-NN y las redes neuronales han sido utilizados para problemas de clasificación. En función de las características poblacionales se clasifica una determinada población en un rango del espectro político. Estos algoritmos han sido ejecutados sobre el conjunto de datos Elecciones al presidente de gobierno en España. Durante todo el trabajo, sobre este conjunto de datos se consideran tres problemas, los tres utilizan las mismas características poblacionales en su ejecución pero se distinguen en las categorías que se consideran a la hora de clasificar.

El primer problema clasifica a cada mesa electoral en una de las siguientes categorías: Izquierda, Centro-izquierda, Centro-derecha, Derecha, Regionalista-izquierda y Regionalista-derecha. Sobre este problema la medida que se ha utilizado para evaluar el rendimiento ha sido la *accuracy*. De ahora en adelante nos referiremos a este problema como *Problema de clasificación 1*.

El segundo problema clasifica a cada mesa electoral en una de las siguientes categorías: Izquierda, Derecha. Sobre este problema la medida que se ha utilizado para evaluar el rendimiento ha sido la *accuracy*, que tiene sentido ya que el número de miembros de cada clase está equilibrado. De ahora en adelante nos referiremos a este problema como *Problema de clasificación 2*.

El tercer problema clasifica a cada mesa electoral en una de las siguientes categorías: Regionalista, No regionalista. Para este problema, con el fin de considerar otra medida hemos supuesto el siguiente escenario. Imaginemos que cierta organización tiene como fin que los partidos regionalistas en España pierdan fuerza. Para ello se van a organizar una serie de mítines en las zonas donde el porcentaje de votos a partidos regionalistas sea alto. El problema es que el presupuesto es muy limitado. Debido a esto el objetivo es que aunque se organicen pocos mítines, estos se realicen en zonas donde realmente los partidos regionalistas tienen fuerza. Dado este escenario, la medida que nos interesa es la *precision*

de la clase Regionalista. Esto es el porcentaje de poblaciones que realmente son regionalistas entre las clasificadas como regionalistas. De ahora en adelante nos referiremos a este problema como *Problema de clasificación 3*.

Ligado a cada uno de estos problemas de clasificación hemos considerado un problema de regresión. En estos problemas las variables dependientes (sobre las que se realizan las regresiones) son el porcentaje de mesas cuyo partido más votado fue de una determinada categoría. Aquí cada muestra, cada “fila” de la tabla se corresponde con una provincia en un determinado año (en un proceso electoral). Esto es diferente al porcentaje de votos de una categoría en una provincia. Al problema de regresión ligado al *Problema de clasificación 1* lo identificaremos como *Problema de regresión 1*. Al problema de regresión ligado al *Problema de clasificación 2* lo identificaremos como *Problema de regresión 2*. Al problema de regresión ligado al *Problema de clasificación 3* lo identificaremos como *Problema de regresión 3*.

3.2.2 Regresión lineal

La Regresión Lineal ha sido utilizada para los dos conjuntos de datos que manejamos en el trabajo. Se ha realizado una regresión lineal utilizando tanto una línea como una parábola para cada problema. Detallaremos la manera de proceder para cada conjunto de datos.

Para el conjunto de datos Elecciones al presidente de gobierno en España, como bien hemos mencionado en la sección anterior, las tablas están diseñadas para resolver problemas de clasificación, no de regresión. Hemos realizado copias de estas tablas y realizado modificaciones para adecuarlas al problema de regresión. Los campos de las características se dejaron tal y como en la tabla principal. El campo categoría (Izquierda, Derecha...) se transformó en varias columnas, cada una tomando el valor del porcentaje de votos a esa categoría sobre el total con las características de la fila correspondiente. Así conseguimos el enfoque numérico necesario para la regresión.

En el caso del conjunto de datos Elecciones al presidente de gobierno en Brasil, Estados Unidos y España las tablas ya están preparadas para una regresión lineal.

Para cada conjunto de datos se ha realizado una regresión lineal mediante línea y mediante parábola por cada rango del espectro político. Por ejemplo, se ha realizado una regresión lineal y mediante parábola con el objetivo de estimar el porcentaje de votos al centro-izquierda tomando como variable independiente el acceso a internet, una regresión lineal y mediante parábola para el centro-derecha tomando como variable independiente la religiosidad... Las regresiones lineales han sido realizadas utilizando una sola variable independiente. Para elegir la característica a utilizar como variable independiente entre todas se ha buscado la característica con más correlación con cada variable que se deseaba estimar. Para encontrar la característica con más correlación se han usado matrices de covarianzas, que mostraremos en el capítulo de *Integración, pruebas y resultados*.

La regresión ha sido implementada en Python utilizando las librerías Sklearn y Matplotlib para graficar. En cada regresión se ha graficado la nube de puntos representando los valores reales, en azul. Los puntos poseen un tamaño acorde al peso que posee ese valor de la variable en la regresión. Se ha graficado en rojo la recta o parábola obtenidas mediante la

regresión. Por otro lado, el NRMSD ha sido hallado en cada regresión. Adjuntamos a continuación el gráfico de un ejemplo de regresión lineal mediante parábola.

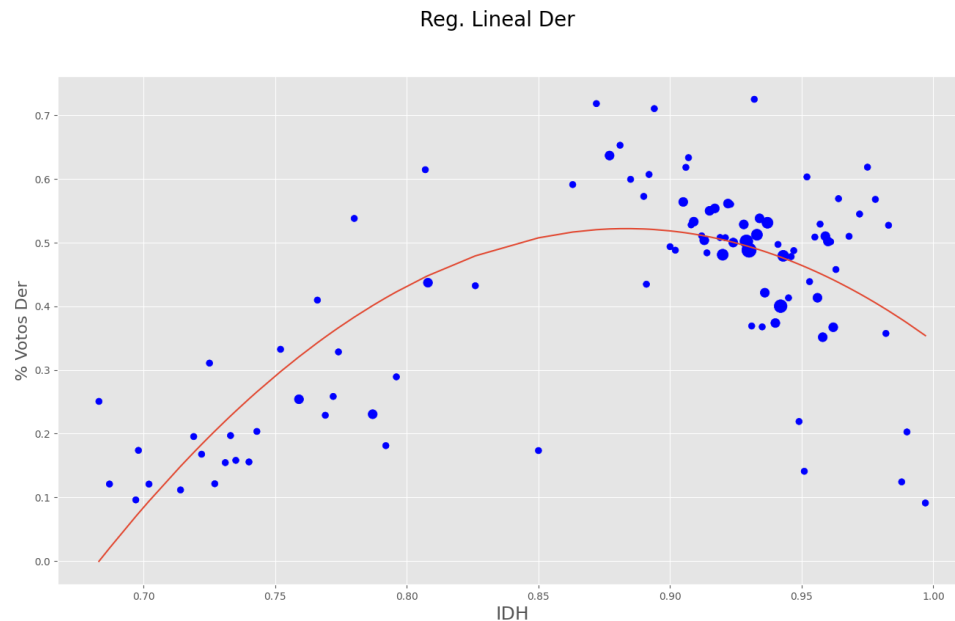


Figura 3-3 Ejemplo de regresión lineal mediante parábola

3.2.3 Algoritmo K-NN

Este algoritmo ha sido utilizado en los problemas de clasificación para el conjunto de datos Elecciones al presidente de gobierno en España. Esto es para los problemas *Problema de clasificación 1*, *Problema de clasificación 2* y *Problema de clasificación 3*. Se han utilizado todas las características recopiladas en la base de datos. Esto quiere decir que cada muestra de la base de datos se transforma en un vector multidimensional (de más de 22 dimensiones) en la fase de entrenamiento del algoritmo.

Para la implementación se ha utilizado Python y la librería Sklearn y el código ha sido ejecutado en Google Colab. Se ha ejecutado este mismo algoritmo para los tres problemas de clasificación. Además, para cada problema se han realizado distintas ejecuciones variando el número de vecinos entre 3, 5, 7 y 9, la distancia utilizada entre Euclidea, Manhattan y Chebyshev, y el porcentaje de datos utilizados para el test entre 30%, 25%, 20%, 15% y 10%. Para cada ejecución se ha obtenido la matriz de confusión. A partir de esta matriz se ha calculado la *accuracy* en los problemas de clasificación 1 y 2 y la *precision* en el tercero.

3.2.4 Red neuronal Single-layer

La primera red neuronal considerada es una red neuronal sin capa intermedia. Esto quiere decir que solo posee una capa de entrada y una capa de salida. Las características numéricas han sido normalizadas antes de ser procesadas por la red neuronal. Las características categóricas, como por ejemplo Gobierno de EE. UU, han sido transformadas a valores numéricos mediante el proceso llamado *One Hot Encoding*. Las categorías (rangos del espectro político) también han sido sometidas al proceso de One Hot Encoding para poder

ser manejadas como valores numéricos. El proceso de normalización y One Hot Encoding son detallados en el capítulo *Desarrollo*.

La capa de entrada está formada por tantas neuronas como características poblacionales consideramos (y las que se añaden tras el proceso de One Hot Encoding). El valor de salida de las neuronas de esta capa es el mismo que el de entrada. Cada neurona de la capa de entrada está conectada con todas las neuronas de la capa de salida.

La capa de salida está formada por tantas neuronas como categorías tenga nuestro problema de clasificación. Por lo tanto en el *Problema de clasificación 1* la capa de salida posee seis neuronas. En los *Problemas de clasificación 2 y 3* la capa de salida posee dos neuronas. La función de activación de las neuronas de esta capa es *Softmax*, la más usada en problemas de clasificación. La salida de nuestra red neuronal es un vector donde cada elemento de este es la probabilidad de que una categoría sea la que más votos haya recibido en la mesa electoral que se haya usado como entrada. La función de pérdida que se ha usado para entrenar la red neuronal es *Categorical crossentropy*.

Se ha aplicado esta red neuronal a cada problema de clasificación. Se han realizado múltiples ejecuciones en las que se ha variado el número de épocas entre 10, 50, 100 y 500, el tamaño de lote entre 10, 32, 62 y 132. El porcentaje de datos utilizados para el test entre 30%, 25%, 20%, 15% y 10%. También se han realizado múltiples ejecuciones variando el método de optimización. Se ha utilizado la función Adam y también RMSProp, que se ha ejecutado para distintos valores de momentum y tasa de aprendizaje.

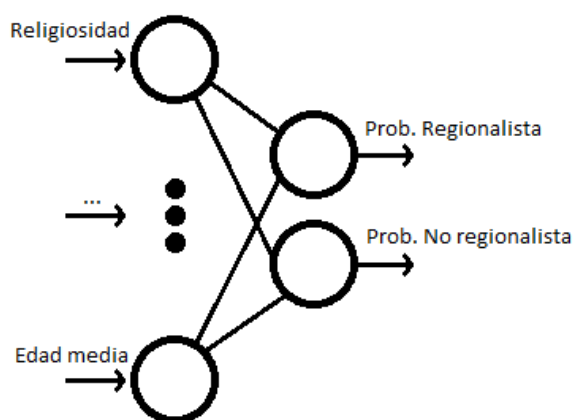


Figura 3-4 Red neuronal de una capa del Problema de clasificación 3

3.2.5 Red neuronal Multi-Layer

La segunda red neuronal considerada en el trabajo es una red neuronal con una capa intermedia. No se han considerado redes neuronales con más capas intermedias ya que añadir capas intermedias en pocos casos mejora el rendimiento de la red neuronal. Del mismo modo que en la red neuronal anterior las características numéricas han sido normalizadas y las características categóricas han sido transformadas a valores numéricos mediante One Hot Encoding. Las categorías también son sometidas al proceso de One Hot Encoding.

La capa de entrada está formada por tantas neuronas como características poblacionales consideramos (y las que se añaden tras el proceso de One Hot Encoding). Por lo tanto el número de neuronas de entrada es el mismo que en la red neuronal sin capa intermedia. El valor de salida de las neuronas de esta capa es el mismo que el de entrada. Cada neurona de la capa de entrada está conectada con todas las neuronas de la capa oculta (intermedia).

La capa oculta posee un número de neuronas mayor que el número de neuronas de la capa de salida pero menor que el número de neuronas de la capa de entrada. La función de activación de las neuronas que se ha usado en esta capa ha sido *ReLU*, aunque se han considerado otras funciones en la fase de variación de parámetros. Cada neurona de la capa intermedia está conectada con todas las neuronas de la capa de salida.

La capa de salida, del mismo modo que en la red neuronal anterior, está formada por tantas neuronas como categorías tenga nuestro problema de clasificación. En el *Problema de clasificación 1* la capa de salida posee 6 neuronas. En los *Problemas de clasificación 2 y 3* la capa de salida posee 2 neuronas. La función de activación de las neuronas de esta capa es *Softmax*, la más usada en problemas de clasificación. La salida de nuestra red neuronal es un vector donde cada elemento de este es la probabilidad de que una categoría sea la que más votos haya recibido en la mesa electoral que se haya usado como entrada. La función de pérdida que se ha usado para entrenar la red neuronal es *Categorical crossentropy*.

Se ha aplicado esta red neuronal a cada problema de clasificación. Se han realizado múltiples ejecuciones en las que se ha variado el número de neuronas de la capa intermedia, la función de activación de la capa intermedia, el número de épocas, el tamaño de lote y el porcentaje de datos utilizados para el test. También se han realizado múltiples ejecuciones variando el método de optimización. Se ha utilizado la función Adam variando la tasa de aprendizaje y también RMSProp, que se ha ejecutado para distintos valores de momentum y tasa de aprendizaje.

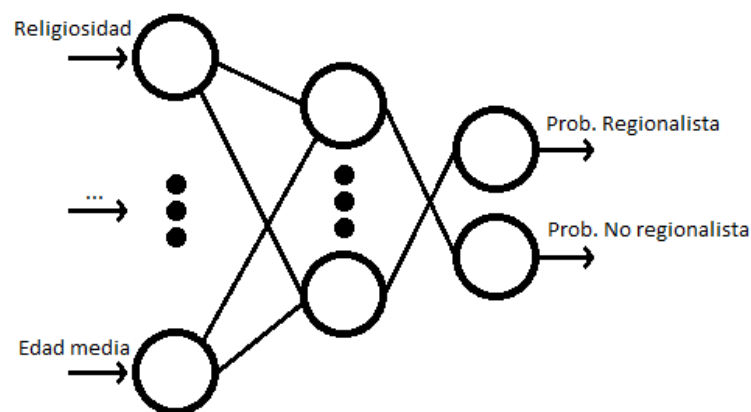


Figura 3-5 Red neuronal multicapa del Problema de clasificación 3

3.3 Flujo de trabajo

Los algoritmos utilizados han sido implementados mediante el lenguaje de programación Python. En primer lugar se escribió un programa básico para cada algoritmo, sin variación

de parámetros. Tras la comprobación del correcto funcionamiento del código y la validación por parte del tutor se elaboraron nuevas versiones de los programas en las que se ejecutan los algoritmos repetidas veces en un bucle. En cada iteración de los bucles hacemos variar los parámetros de los algoritmos. En el caso de K-NN por ejemplo variamos el número de vecinos, en las redes neuronales el número de épocas, el número de neuronas de la capa intermedia... Las medidas adecuadas al problema que se esté tratando son obtenidas en cada ejecución de cada algoritmo. Tras esto los resultados obtenidos son ordenados y se realizan comparaciones entre distintas versiones de un mismo algoritmo y entre distintos algoritmos (aplicados al mismo problema). Estos resultados y comparaciones los mostramos en el capítulo *Integración, pruebas y resultados*.

4 Desarrollo

4.1 Recopilación de los datos

En esta sección comentamos estas fuentes y el proceso de recopilación de datos. Los datos electorales relativos a Brasil y Estados Unidos han sido obtenidos desde las webs <http://www.tse.jus.br/eleicoes/estatisticas/estatisticas-eleitorais> y <https://dataverse.harvard.edu/dataverse/eda> respectivamente. Los datos relativos a España han sido obtenidos desde la página web del MIR, <http://www.infoelectoral.mir.es/infoelectoral/min/areaDescarga.html?method=inicio>. De aquí se han descargado todos los datos relativos a los procesos electorales, es decir, el año, la comunidad autónoma y provincia de cada mesa electoral y cuál fue el partido político más votado en cada mesa. A continuación mostramos una tabla que detalla las URLs desde las que se han obtenido los datos relativos a cada característica poblacional considerada. La tabla que identifica cada característica con su identificador numérico puede ser encontrada en el anexo Identificadores numéricos características.

Tabla 4-1 Fuentes base de datos

Característica	Fuente
1	https://blog.iese.edu/martinezabascal/2016/04/28/que-provincia-es-mas-catolica-y-2/
2	https://www.ine.es/jaxiT3/Tabla.htm?t=3199
3	https://www.ine.es/jaxiT3/Tabla.htm?t=6369
4	https://www.ine.es/jaxiT3/Tabla.htm?t=1485
5	Wikipedia
6	Wikipedia
7	Wikipedia
8	https://www.ine.es/jaxi/Tabla.htm?path=/t20/e244/avance/p02/I0/&file=1mun00.px&L=0
9	https://es.wikipedia.org/wiki/Anexo:Provincias_de_Espa%C3%B1a_por_IDH
10	https://avancedigital.gob.es/banda-ancha/coertura/Documents/Cobertura-BA-2018.pdf
11	INE
12	https://estadisticasdecriminalidad.ses.mir.es/dynPx/inebase/index.htm?type=pcaxis&path=/DatosBalanceAnt/&file=pcaxis
13	https://www.ine.es/jaxi/Tabla.htm?path=/t20/e245/p08/&file=03005.px&L=0
14	https://www.ine.es/jaxi/Tabla.htm?path=/t20/e245/p08/&file=03005.px&L=0
15	https://www.ine.es/jaxiT3/Tabla.htm?t=2074
16	https://www.ine.es/jaxi/Tabla.htm?path=/t20/e244/avance/p02/I0/&file=1mun00.px&L=0
17	https://www.businessintelligence.info/variros/longitud-latitud-pueblos-espana.html
18	https://www.businessintelligence.info/variros/longitud-latitud-pueblos-espana.html
19	https://www.businessintelligence.info/variros/longitud-latitud-pueblos-espana.html
20	http://www.infoelectoral.mir.es/infoelectoral/min/areaDescarga.html?method=inicio

Los formatos en los que se han obtenido los datos han sido CSV, TXT, DAT y formatos nativos de Microsoft Excel.

Los datos relativos a los habitantes por municipio no pudieron ser obtenidos en su totalidad desde la web del INE. Para completar la información se realizó un script de web scrapping para hacer búsquedas en Google automáticamente y obtener los datos. El código de este script puede ser encontrado en el anexo Script Python para la obtención del número de habitantes de cada municipio.

4.2 Construcción de la base de datos

4.2.1 Preprocesamiento de los datos

Los datos, al ser obtenidos de distintas fuentes y en diferentes formatos, necesitaron un preprocesamiento antes de poder ser insertados en una base de datos. Destaca el preprocesamiento necesario para la información electoral. Los datos relativos a cada proceso electoral constaban de varios archivos DAT y un archivo PDF explicando qué información contenía cada documento y cómo se organizaba. Este archivo PDF fue leído en detalle para poder obtener los datos deseados. Tras esto se utilizaron scripts escritos en Python para leer línea por línea los ficheros DAT originales y crear unos segundos archivos con los datos ya organizados de la manera deseada, listos para ser insertados en una base de datos SQL. Para la información relativa a las características poblacionales se siguió el mismo modo de actuar. Los archivos descargados no siempre estaban en un formato adecuado para insertar en una base de datos SQL y fue necesario pasar esos archivos por scripts de Python con el fin de pasar todo a un formato común. Un ejemplo de script para organizar los datos de una forma adecuada es el siguiente

```
with open('pipercapita_parsed.dat', 'r', encoding="utf-8") as f:
    for linea in f:
        campos = linea.split('\t')
        provincia = campos[0]
        for i in range(1,19):
            anno = '20' + str(i-1).zfill(2)

            print(provincia+'\t'+anno+'\t'+campos[i].replace('\n',''))
```

Como se ha comentado en los capítulos de *Introducción* y *Diseño*, cada división territorial posee unos partidos políticos diferentes. Debido a esto, para trabajar en un plano común en los diferentes territorios, los partidos políticos fueron divididos en rangos del espectro político. Para clasificar un partido en izquierda, derecha, regionalista... se ha utilizado el campo “Posición” del artículo de Wikipedia de cada partido político. El detalle de cómo ha sido clasificado cada partido político puede ser encontrado en el anexo Clasificación de los partidos políticos.

4.2.2 Construcción de la base de datos PostgreSQL

Como se ha aclarado en la sección anterior, tras el preproceso mediante scripts de Python, los datos fueron almacenados en archivos de extensión DAT. Estos archivos fueron cargados a tablas PostgreSQL almacenadas localmente. Para realizar esta carga se utilizó la interfaz

gráfica del software PgAdmin, que fue usado también para la creación de la base de datos y de las tablas.

Muchas de las estadísticas obtenidas durante el trabajo como medias, varianzas, covarianzas... han sido adquiridas mediante funciones de PostgreSQL en combinación con scripts de Python. También PostgreSQL ha sido utilizado para crear las tablas temporales necesarias para la creación de las tablas principales y tablas para realizar las regresiones lineales, ya que la tabla principal está diseñada para problemas de clasificación, no de regresión. Por último las tablas almacenadas en PostgreSQL fueron exportadas a archivos CSV, formato adecuado para ser leído por las librerías utilizadas en el código.

4.3 Implementación del código

4.3.1 Python

El lenguaje de programación Python es un lenguaje interpretado, multiplataforma y multiparadigma. Es dinámico y está diseñado para soportar programación orientada a objetos. En menor medida también soporta programación funcional. La sencillez y la legibilidad del código son dos de los aspectos más destacables de este lenguaje de programación.

El lenguaje de programación más utilizado en el trabajo ha sido Python. El código necesario para ejecutar los algoritmos de aprendizaje automático implementados ha sido escrito en este lenguaje. Este código puede ser encontrado en el anexo Código Python. Además, también se han escrito en Python los scripts de preprocesamiento de los datos y los scripts utilizados para realizar gráficas, en concreto, mediante la librería Matplotlib. La versión de Python que se ha usado en los programas que se han ejecutado en local ha sido la 3.8.2. La versión usada en los programas ejecutados en Google Colab ha sido la 3.6.9. La decisión de utilizar este lenguaje de programación se ha tomado por los siguientes motivos. En primer lugar la existencia de librerías extensas y con renombre específicas para manejo de datos, aprendizaje automático y redes neuronales tales como Pandas, Keras, Sklearn y Tensorflow. En segundo lugar por la sencillez del código y la facilidad de implementación que convive con un rendimiento y velocidad de ejecución más que aceptables para el trabajo realizado. Por último, la posibilidad de ejecutar el código en la nube mediante Google Colab, que aporta fluidez en la comunicación entre tutor y alumno.

4.3.2 Google Colab

Un *Jupyter Notebook* es un documento JSON que contiene una lista ordenada de celdas de entradas y salidas que pueden contener código Julia, Python o R, texto, texto matemático y gráficas, entre otras. Puede ser visualizado vía web, en PDF, en LaTeX... Colaboratory, más conocido como *Google Colab*, es un Jupyter Notebook gratuito ofrecido por Google que se ejecuta en la nube y cuyos Notebooks son almacenados mediante Google Drive. Cada cuaderno está conectado a una máquina virtual de *Google Compute Engine* (máquinas virtuales de Google en la nube). Contamos con 12 GB de RAM y 50 GB de almacenamiento en disco.

El entorno de Google Colab es un entorno adecuado para el aprendizaje automático y para nuestro trabajo ya que es capaz de ejecutar tanto Python 2 como Python 3 y además Google nos da la opción de utilizar sus GPU y TPU en lugar de CPU para realizar ejecuciones más potentes. Además, por defecto cuenta con varias librerías de aprendizaje automático y manejo de datos instaladas, tales como Pandas, Keras, Sklearn y Tensorflow, que son las que usamos en nuestro trabajo. Si fuera necesario, es posible instalar cualquier librería Python en nuestra máquina virtual.

Para el desarrollo del trabajo ha sido una herramienta útil no solo por estas características, si no porque además facilita la compartición de código entre varios usuarios, en este caso alumno y tutor, lo que nos ha permitido más dinamismo a la hora de realizar pruebas. Además, ha eliminado los posibles contratiempos que pueden generar la discrepancia en los resultados generada por la ejecución del mismo código en entornos distintos.

4.3.3 Pandas, Keras, Sklearn y Tensorflow

Las librerías de Python más destacadas para la realización del trabajo han sido Pandas, Keras, Sklearn y Tensorflow.

La biblioteca Pandas es una extensión de la librería NumPy de Python creada por Wes McKinney en 2008. La finalidad de esta librería es la manipulación y análisis de datos. Utiliza un tipo de dato llamado DataFrame para manipular datos que posee indexación integrada. En el trabajo ha sido utilizada debido a que funciones de Sklearn que se utilizan toman como entrada DataFrames. En concreto, los datos almacenados en la base de datos PostgreSQL son transformados a un DataFrame que se pasa a la función `carga_datos_csv` descrita en el anexo Función `carga_datos_csv`.

Esta función es de vital importancia en la realización de nuestro trabajo ya que se usa en todos los programas que ejecutan el Algoritmo KNN y las redes neuronales.

Scikit-learn (Sklearn) es una biblioteca de aprendizaje automático para el lenguaje de programación Python que empezó como un proyecto de Google Summer of Code por David Cournapeau. Posee algoritmos de clasificación y regresión. Está pensada para interactuar con las estructuras de datos propias de NumPy, SciPy y Pandas. En nuestro trabajo es usada en el preprocesamiento de los datos, en concreto dentro de la función `carga_datos_csv` se llama a la función de Sklearn `train_test_split` que realiza la separación entre el conjunto de datos de test y conjunto de datos de entrenamiento. También esta librería posee un papel importante ya que el algoritmo de clasificación KNN y las regresiones lineales se realizan utilizando las clases `KNeighborsClassifier` y `linear_model` de esta librería.

Keras es una librería de redes neuronales de código abierto escrita en Python cuyo autor principal es el ingeniero de Google François Chollet. Keras posee diversas implementaciones de los bloques que componen usualmente las redes neuronales, como por ejemplo sus capas. Esta biblioteca es capaz de ejecutarse sobre *TensorFlow*, que es una biblioteca (de Python entre otros lenguajes) desarrollada por Google Brain desarrollada para el aprendizaje automático. En nuestro trabajo se ha importado Keras desde TensorFlow y se ha utilizado para la creación de los modelos de Redes Neuronales, tanto la red neuronal sin capas ocultas como la red neuronal con una capa oculta.

5 Integración, pruebas y resultados

5.1 Estudio estadístico básico

Antes de ejecutar cualquier algoritmo de regresión o de clasificación sobre nuestro conjunto de datos, hemos realizado un estudio estadístico básico ligado a cada problema de regresión considerado. Hemos hallado las matrices de covarianzas, las medias, varianzas, máximos y mínimos de las variables dependientes. Del mismo modo que en los problemas de clasificación, sobre las variables categóricas se aplicó One Hot Encoding y se normalizaron todas las variables.

En la Tabla 5-1 mostramos la media, varianza, máximos y mínimos de cada variable dependiente del *Problema de regresión 1* (de esta tabla se pueden derivar estos mismos estadísticos para los otros problemas de regresión). En la Tabla 5-2 mostramos las variables categóricas y no categóricas con más correlación con cada variable dependiente del *Problemas de regresión 1*. En el anexo Matrices de covarianzas, varianzas, medias, máximos y mínimos mostramos estas mismas tablas para los *Problemas de regresión 2, 3, 4*. También en este anexo presentamos las medias, varianzas, máximos y mínimos sujetos a diferentes restricciones y las matrices de covarianzas completas. Por ejemplo, la media sujeta a cierta provincia, o sujeta a un proceso electoral determinado.

Tabla 5-1 Media, varianza, máximos y mínimos Elecciones gobierno de España

Variable	Media	Máximo	Tiempo y lugar máximo	Mínimo	Tiempo y lugar mínimo	Varianza
Regionalista-izquierda	0,0312	0,8206	2019/4 Lleida	0,0000	Varias muestras	0,01384
Regionalista-derecha	0,0465	0,8945	2011/11 Lleida	0,0000	Varias muestras	0,01895
Centro-izquierda	0,3516	0,8863	2019/4 Jaén	0,0000	Varias muestras	0,06361
Centro-derecha	0,0065	0,1572	2019/4 Madrid	0,0000	Varias muestras	0,00039
Izquierda	0,0328	0,6715	2015/12 Barcelona	0,0000	Varias muestras	0,00963
Derecha	0,5314	0,9920	2011/11 Murcia	0,0000	Varias muestras	0,09122

Tabla 5-2 Variables con más correlación Problema de regresión 1

Variable	Más correlación	Valor	Menos correlación	Valor
Regionalista-izquierda	comunidad_cataluña	0,3718	provincia_araba_Álava	0,0011
Regionalista-derecha	gobierno_comunidad_region_der	0,3736	provincia_teruel	-0,0018
Centro-izquierda	gobierno_usa_rep	0,3577	provincia_león	0,0015
Centro-derecha	anno	0,0979	provincia_huelva	-0,0001
Izquierda	gobierno_comunidad_region_der	0,2121	provincia_coruña_a	0,0000

Derecha	comunidad_cataluña	-0,3595	provincia_asturias	-0,0011
Variable	No categórica más correlación	Valor	No categórica menos correlación	Valor
Regionalista-izquierda	religiosidad	-0,2882	inmigracion_eur	-0,0114
Regionalista-derecha	religiosidad	-0,3013	edad_media	0,0090
Centro-izquierda	internet	0,0434	viajeros	-0,0092
Centro-derecha	anno	0,0979	longitud	0,0068
Izquierda	religiosidad	-0,1863	habitantes_municipio	0,0238
Derecha	religiosidad	0,2902	edad_media	0,0060

De la Tabla 5-1 destacamos que el 99,2 % de las mesas electorales en Murcia tuvieron un partido de derecha como partido más votado en 2011. También destaca que la derecha es la categoría que tiene media más alta. Merece la pena explicar por qué todos los mínimos son 0 %. Esto es debido a que el 0 % no significa que en una provincia no hubo ningún voto a una categoría determinada, si no que en ninguna mesa el partido más votado fue de dicha categoría.

En las tablas anteriores podemos observar que en Estados Unidos y España las medias de izquierda y derecha son más altas que en Brasil. Esto es debido a que en Estados Unidos y en España el voto está más acumulado en los cinco partidos principales (de los que proceden los datos), sin embargo, en Brasil se divide el voto entre más partidos, por eso las medias son más bajas. Destaca también el District of Columbia, que claramente es la unidad territorial considerada que más vota a la izquierda y además la que menos vota a la derecha.

5.2 Regresión lineal

En la sección anterior se han hallado las variables más correlacionadas con las variables dependientes en cada problema de regresión. Hemos utilizado estas variables para realizar una regresión lineal usando una línea y una parábola para cada variable dependiente. En cada regresión se ha hallado el NRMSE (explicado en la sección Medidas) y se ha graficado la recta o parábola conseguida frente a la nube de puntos que representa los datos reales.

5.2.1 Regresión lineal mediante línea

Se ha realizado una regresión lineal utilizando una línea para estimar cada una de las variables dependientes en cada uno de los problemas de regresión considerados. Todos los gráficos de las regresiones lineales pueden ser encontrados en el anexo Gráficos de las regresiones lineales. A continuación adjuntamos una tabla por cada problema de regresión que muestra las regresiones lineales mediante línea realizadas ordenadas según su NRMSE de mayor a menor, es decir, ordenadas según lo “mala” que ha sido la regresión.

Tabla 5-3 Resultados regresión lineal mediante línea Problema de regresión 4

Variable dependiente	Variable independiente	NRMSE	Varianza
Derecha Estados Unidos	IDH	0,2260	0,1977

Izquierda Estados Unidos	IDH	0,2253	0,1420
Derecha España	IDH	0,2250	0,1277
Derecha Brasil	IDH	0,2247	0,3019
Derecha todos	IDH	0,2181	0,3244
Izquierda Brasil	IDH	0,2133	0,1546
Izquierda España	IDH	0,2000	0,0162
Izquierda todos	IDH	0,1834	0,4016

Como podemos ver la variable dependiente para la que se ha hecho una peor predicción en el *Problema de regresión 4* es el porcentaje de votos a la derecha en Estados Unidos, para la que mejor la Izquierda en el conjunto de los tres países. Destaca la poca varianza del porcentaje de votos a Izquierda en España respecto al IDH.

Tabla 5-4 Resultados regresión lineal mediante línea Problema de regresión 1

Variable dependiente	Variable independiente	NRMSE	Varianza
Centro-derecha	Año	0,2594	0,4894
Regionalista-derecha	Religiosidad	0,2537	0,2935
Centro-izquierda	Internet	0,2435	-0,0172
Regionalista-izquierda	Religiosidad	0,2430	0,2094
Derecha	Religiosidad	0,2206	0,2690
Izquierda	Religiosidad	0,1686	0,6909

La variable dependiente para la que se ha hecho una peor predicción en el *Problema de regresión 1* es el porcentaje de votos al Centro-derecha, para la que mejor la Izquierda.

Tabla 5-5 Resultados regresión lineal mediante línea Problema de regresión 2

Variable dependiente	Variable independiente	NRMSE	Varianza
Izquierda	Año	0,2974	0,0754
Derecha	Criminalidad	0,2634	0,0470

Como vemos los resultados son ligeramente peores que en las anteriores regresiones. La regresión para la izquierda presenta peores resultados que la regresión para la derecha.

Tabla 5-6 Resultados regresión lineal mediante línea Problema de regresión 3

Variable dependiente	Variable independiente	NRMSE	Varianza
Regionalista	Religiosidad	0,2443	0,3273
No regionalista	Religiosidad	0,2443	0,3273

Vemos que los resultados de NRMSE y Varianza son los mismos, esto tiene sentido ya que sólo existen dos clases y se ha utilizado la misma variable independientes para realizar la regresión.

5.2.2 Regresión lineal mediante parábola

Además de la regresión lineal mediante línea se ha realizado una regresión lineal utilizando una parábola. Todos los gráficos de las regresiones lineales mediante parábola pueden ser encontrados en el anexo Gráficos de las regresiones lineales. A continuación adjuntamos una tabla por cada problema de regresión que muestra las regresiones lineales mediante parábola realizadas ordenadas según su NRMSE de mayor a menor, es decir, ordenadas según lo “mala” que ha sido la regresión.

Tabla 5-7 Resultados regresión lineal mediante parábola Problema de regresión 4

Variable dependiente	Variable independiente	NRMSE	Varianza
Derecha Brasil	IDH	0,2194	0,3345
Derecha España	IDH	0,2171	0,1878
Izquierda Brasil	IDH	0,2133	0,1543
Izquierda Estados Unidos	IDH	0,1986	0,3329
Derecha Estados Unidos	IDH	0,1963	0,3948
Izquierda todos	IDH	0,1834	0,4015
Derecha todos	IDH	0,1793	0,5432
Izquierda España	IDH	0,1671	0,3135

Vemos que ahora la variable que mejores resultados presenta ha cambiado respecto a la regresión mediante línea, ahora es la Izquierda en España.

Tabla 5-8 Resultados regresión lineal mediante parábola Problema de regresión 1

Variable dependiente	Variable independiente	NRMSE	Varianza
Centro-izquierda	Internet	0,2369	0,0374
Regionalista-derecha	Religiosidad	0,2189	0,4741
Regionalista-izquierda	Religiosidad	0,2001	0,4638
Derecha	Religiosidad	0,1964	0,4205
Centro-derecha	Año	0,1735	0,7716
Izquierda	Religiosidad	0,1669	0,6971

Vemos que al realizar la regresión mediante parábola el orden de los NRMSE de cada variable también ha cambiado en este problema de regresión.

Tabla 5-9 Resultados regresión lineal mediante parábola Problema de regresión 2

Variable dependiente	Variable independiente	NRMSE	Varianza
Izquierda	Año	0,2816	0,1713
Derecha	Criminalidad	0,2612	-0,0300

Comprobamos que aquí el orden de los NRMSE es el mismo que en la regresión mediante línea. Además, destaca que los valores de NRMSE prácticamente no han cambiado respecto de la anterior regresión.

Tabla 5-10 Resultados regresión lineal mediante parábola Problema de regresión 3

Variable dependiente	Variable independiente	NRMSE	Varianza
Regionalista	Religiosidad	0,1862	0,6095
No regionalista	Religiosidad	0,1862	0,6095

Otra vez vemos que los valores de NRMSE son los mismos en ambas clases al utilizar la misma variable independiente para realizar la regresión.

Como es obvio los valores de NRMSE son más bajos para la parábola que para la línea, es decir, las predicciones utilizando parábola son mejores.

5.3 Algoritmo K-NN

El algoritmo K-NN ha sido utilizado en los tres problemas de clasificación considerados. Como ya hemos explicado en la sección Problemas considerados, para los dos primeros problemas la medida utilizada ha sido la *accuracy* mientras que para el último ha sido la *precision*. Hemos ejecutado este algoritmo variando el número de vecinos, la distancia utilizada y el porcentaje de datos usado para el test. Hemos hallado la matriz de confusión y a partir de esta la *accuracy* y *precision* para cada combinación de parámetros. Mostramos a continuación un gráfico por cada problema de clasificación que muestra los resultados para cada combinación de parámetros. En el anexo Tablas resultados Algoritmo KNN mostramos las tablas que contienen los datos que se han graficado en estas figuras ordenadas de mejor a peor.

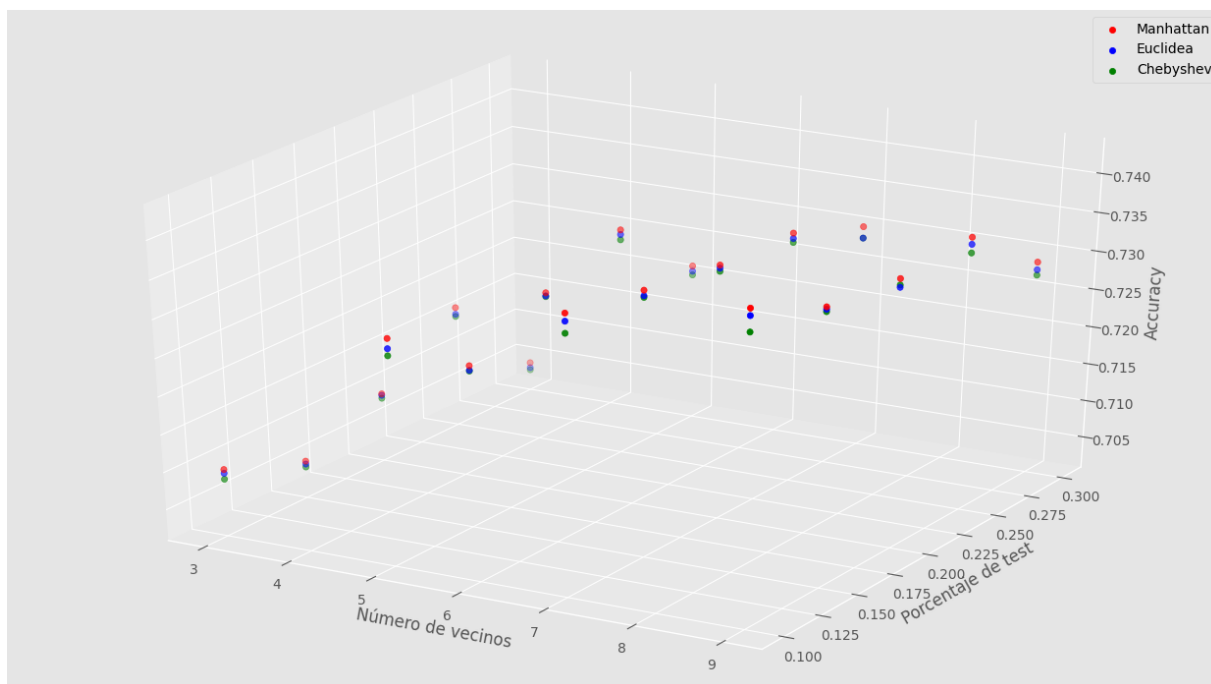


Figura 5-1 Resultados KNN Problema de clasificación 1

Vemos que en el *Problema de clasificación 1* la tendencia es a mejorar la *accuracy* según sube el número de vecinos y aumenta el porcentaje de datos usados para la fase de entrenamiento. La distancia que mejores resultados ha mostrado es Manhattan.

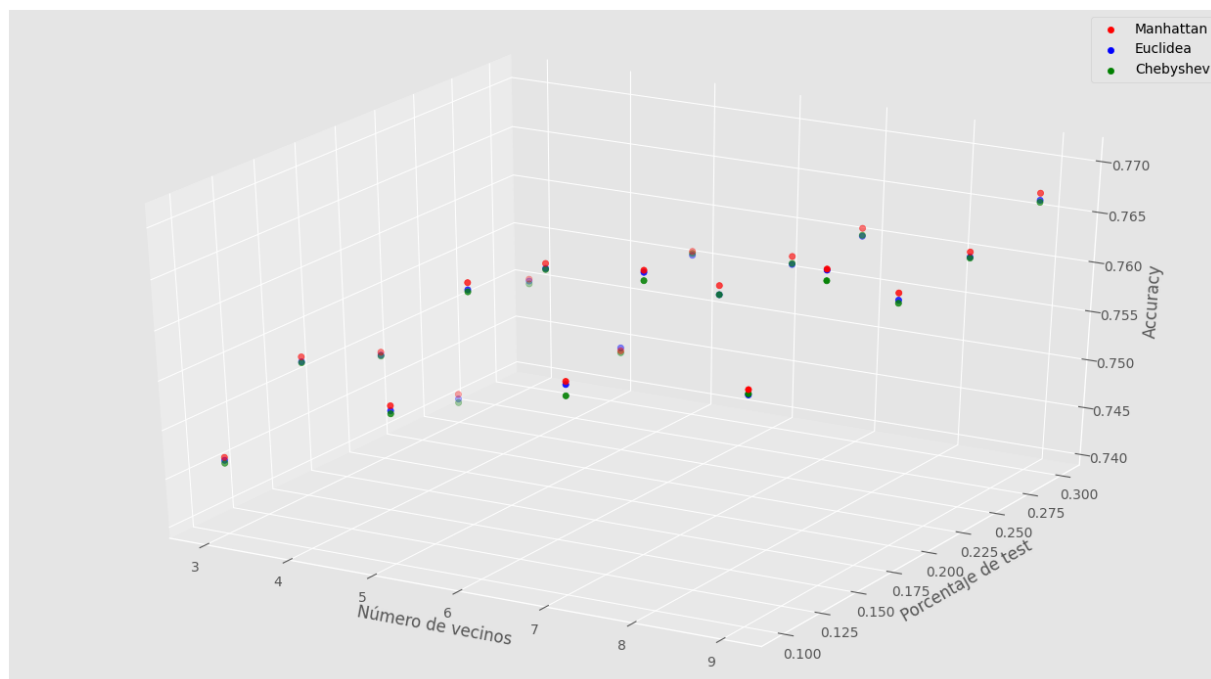


Figura 5-2 Resultados KNN Problema de clasificación 2

En el *Problema de clasificación 2* se repiten las observaciones realizadas para el 1. La mejor distancia vuelve a ser Manhattan y cuantos más vecinos mejor es la predicción. En este caso

el mejor porcentaje para test sigue siendo bajo, pero 0,15 da mejores resultados que 0,10. El nivel de *accuracy* alcanzado en este problema es ligeramente mayor que en el anterior.

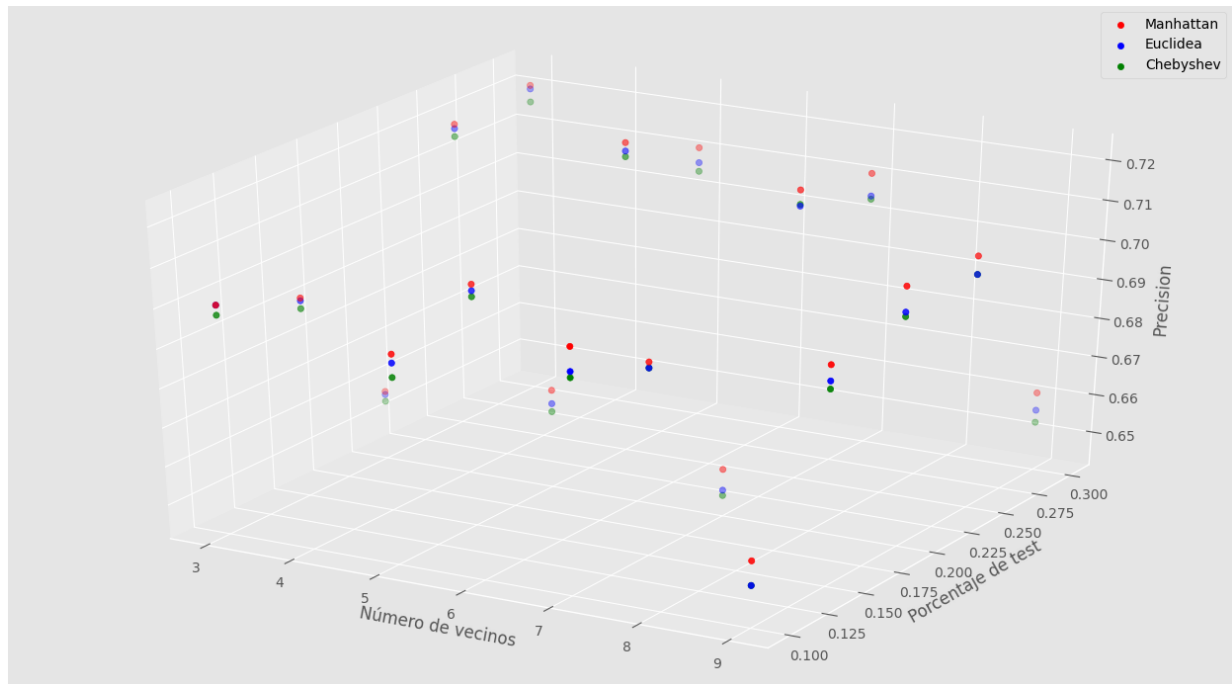


Figura 5-3 Resultados KNN Problema de clasificación 3

Manhattan vuelve a ser la mejor distancia en el *Problema de clasificación 3*. Sin embargo, a diferencia de en los otros problemas, un menor número de vecinos muestra un mejor comportamiento. Esto puede deberse a que para este último problema medimos la *precision*, no la *accuracy*.

5.4 Red neuronal Single-Layer

Como se ha explicado en el capítulo *Diseño*, se han implementado redes neuronales sin capa oculta para cada problema de clasificación considerado. Comentamos a continuación los resultados obtenidos para cada combinación de parámetros para cada problema de clasificación.

Para evitar el *overfitting* se ha comparado las métricas obtenidas para el conjunto de entrenamiento y el conjunto de test. Si el modelo sufre *overfitting* las medidas obtenidas para el conjunto de entrenamiento mejorarán sin cesar mientras que las medidas obtenidas para el conjunto de test empezarán a empeorar a partir de cierto valor. A continuación, mostramos la *accuracy* obtenida para el *Problema de clasificación 1* para el conjunto de test y entrenamiento en función del número de épocas.

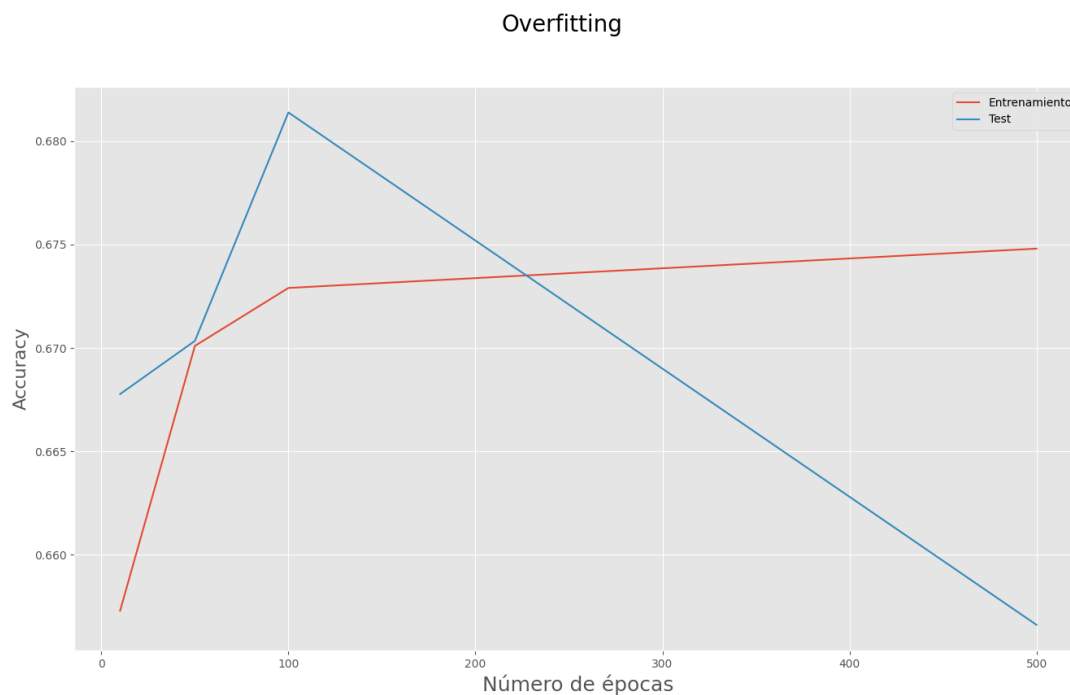


Figura 5-4 Overfitting

Como vemos entre el valor 10 y 100 del número de épocas el valor de la *accuracy* es prácticamente el mismo en los conjuntos de test y de entrenamiento (o incluso superior en el conjunto de test). Sin embargo, para valores más grandes se produce *overfitting*, la *accuracy* del conjunto de entrenamiento sigue mejorando y sin embargo la del conjunto de test disminuye.

En el anexo Resultados Red neuronal Single-layer encontramos el resto de resultados obtenidos realizando la variación de parámetros para cada problema de clasificación. En el *Problema de clasificación 1* se han obtenido mejores resultados utilizando el método de optimización Adam que RMSProp. El número de épocas ideal es 100. El valor de la *accuracy* en función del tamaño de lote se mantiene prácticamente constante a partir de 32, aunque el mejor valor se ha obtenido para 132. El valor ideal de porcentaje de test ha sido 0,15. El mejor valor de *accuracy* obtenido ha sido 0,6879. En el *Problema de clasificación 2* se han obtenido resultados similares utilizando el método de optimización Adam que RMSProp. El número de épocas ideal para Adam ha sido 100. El valor de tamaño de lote ideal ha sido 62. El mejor valor de porcentaje de test ha sido 0,2. Mediante la función RMSProp el mejor valor de *accuracy* ha sido 0,71662 para una tasa de aprendizaje de 0,1 y un momentum de 0. El mejor valor de *accuracy* obtenido con Adam ha sido 0,71933. Recordamos que para el *Problema de clasificación 3* la medida que estamos teniendo en cuenta es la *precision*. La *accuracy* obtenida para los distintos parámetros en casi todos los casos ha sido mayor de 0,9. Esto es debido a que hay muy pocos registros de clase No regionalista. Sin embargo, la *precision* obtenida ha sido muy mala, de 0 en casi todos los casos excepto en pocas combinaciones que ha sido de 1. En los casos en los que la precisión ha sido 1 la red neuronal a lo sumo ha clasificado una muestra como No regionalista, si esta única clasificación la ha realizado de manera correcta el valor de la *precision* ha sido 1, si no, 0. En general podemos decir que el funcionamiento de este algoritmo en este problema ha sido muy malo.

5.5 Red neuronal Multi-Layer

Tal y como describimos en el capítulo *Diseño*, se han implementado Redes neuronales con una capa oculta para cada problema de clasificación considerado. Comentamos a continuación los resultados obtenidos para cada combinación de parámetros para cada problema de clasificación.

El primer hecho a destacar es la función de activación elegida para la capa de neuronas intermedia. En primer lugar se realizaron pruebas utilizando la función Sigmoide o la Tangente hiperbólica. El algoritmo no llegaba a converger y se mantenía en un valor de *accuracy* constante (bastante malo). Se cambió la función de activación a ReLU y los resultados mejoraron notoriamente. El valor de *accuracy* obtenido para el *Problema de Clasificación 1* con método de optimización *Adam* y 71 neuronas en la capa intermedia con la función de activación Sigmoide fue de 0,46242 mientras que con la misma combinación de parámetros pero utilizando ReLU obtuvimos 0,68278, que es notablemente mejor. Para el resto de ejecuciones para los diferentes problemas de clasificación se utilizó por lo tanto ReLU.

En el anexo Resultados Red neuronal Multi-layer encontramos todos los resultados obtenidos realizando la variación de parámetros para cada problema de clasificación. En las diferentes ejecuciones hemos variado la función de activación de la capa intermedia, el número de neuronas de la capa intermedia, el número de épocas, el tamaño de lote, el porcentaje utilizado para el conjunto de test, el método de optimización y su tasa de aprendizaje y momentum. En el *Problema de clasificación 1* los resultados obtenidos para un número de neuronas menor que 25 han sido bastante malos, con una media de 0,46388 de *accuracy*. A partir de 71 neuronas en la capa intermedia el valor de la *accuracy* se ha estabilizado en torno a 0,68. La *accuracy* se estabilizada a partir de un número de épocas de 100, para el que se ha obtenido un valor de 0,68097. La tasa de aprendizaje ideal para Adam de entre las utilizadas ha sido 0,0001, para el que se ha obtenido un valor de nuestra medida de 0,68852. El mejor tamaño de lote ha sido 32, para el que se ha obtenido un valor de *accuracy* de 0,68517. Para un valor menor, como 10, la *accuracy* baja a 0,62439. El mejor resultado obtenido variando el porcentaje de test ha sido 0,69551, mediante un 10% de los datos para realizar el test. Los resultados utilizando RMSProp en lugar de Adam son peores. Para una tasa de aprendizaje mayor que 0,00001 el algoritmo parece no converger y la *accuracy* alcanzada se estanca en aproximadamente 0,46. El mejor resultado para este método de optimización es una *accuracy* de 0,66121, obtenida mediante una tasa de aprendizaje de 0,00001 y un momentum de 0,99. A continuación comentamos los resultados obtenidos para el *Problema de clasificación 2*. Al variar el número de neuronas hemos obtenido en ocasiones valores de *accuracy* en torno a 0,52 y otras veces en torno a 0,71. Según aumentábamos el número de neuronas había menos resultados en torno a 0,52, pero seguían apareciendo, indicando que el algoritmo en ocasiones no convergía. El mejor resultado obtenido ha sido 0,71618 para 92 neuronas (el mayor número de neuronas de nuestra prueba). Para evitar que el algoritmo en ocasiones no convergiera hemos variado la tasa de aprendizaje de Adam y el resultado ideal ha sido para un valor de 0,0001, con el que hemos obtenido 0,72070 de *accuracy*. En las ejecuciones siguientes hemos utilizado este valor de tasa de aprendizaje. Al variar el número de épocas los resultados han sido parejos, pero mejor cuantas más épocas. El valor de *accuracy* más alto ha sido para 500 épocas, de 0,73962. En las siguientes ejecuciones hemos utilizado un número de épocas de 100, con el que se obtenía un valor de 0,73, muy semejante. Los tamaños de lote pequeños han funcionado mejor que los más grandes. El mejor ha sido un tamaño de 32 con el que se

obtenía un valor de 0,72472 de *accuracy*. El porcentaje de test que mejor ha funcionado ha sido 0,1, con el que hemos conseguido un valor de 0,72640 de *accuracy*. A pesar de esto, en las siguientes ejecuciones hemos usado un valor de 0,2, ya que el resultado obtenido era prácticamente idéntico y así prevenimos el overfitting. Al cambiar el método de optimización a RMSProp hemos obtenido mucho peores resultados. En ninguna de las combinaciones de tasa de aprendizaje y momentum para este método se ha obtenido una *accuracy* mayor a 0,53. Los resultados para el *Problema de clasificación 3* han sido muy similares a los obtenidos mediante la Red neuronal Single-Layer. En casi todas las ejecuciones para cualquier combinación de parámetros no se ha predicho ninguna muestra como perteneciente a la clase No regionalista. Para la ejecución con tamaño de lote 62 se ha clasificado una muestra como No regionalista y esta clasificación fue correcta, alcanzando una precisión de 1 en este caso. El algoritmo ha funcionado muy mal para este problema de clasificación. Por último, con el objetivo de hacer más visual esta variación de parámetros, adjuntamos una gráfica que muestra la *accuracy* obtenida para las distintas tasas de aprendizaje (eje en escala logarítmica). Los puntos que se muestran son las *accuracys* mencionadas en este párrafo. Los valores en rojo hacen referencia al *Problema de clasificación 1*, los valores en azul hacen referencia al *Problema de clasificación 2*. Los puntos unidos mediante líneas son ejecuciones con exactamente los mismos parámetros, variando solamente la tasa de aprendizaje. Se ve claramente una tendencia a mejorar la *accuracy* según decrece la tasa de aprendizaje.

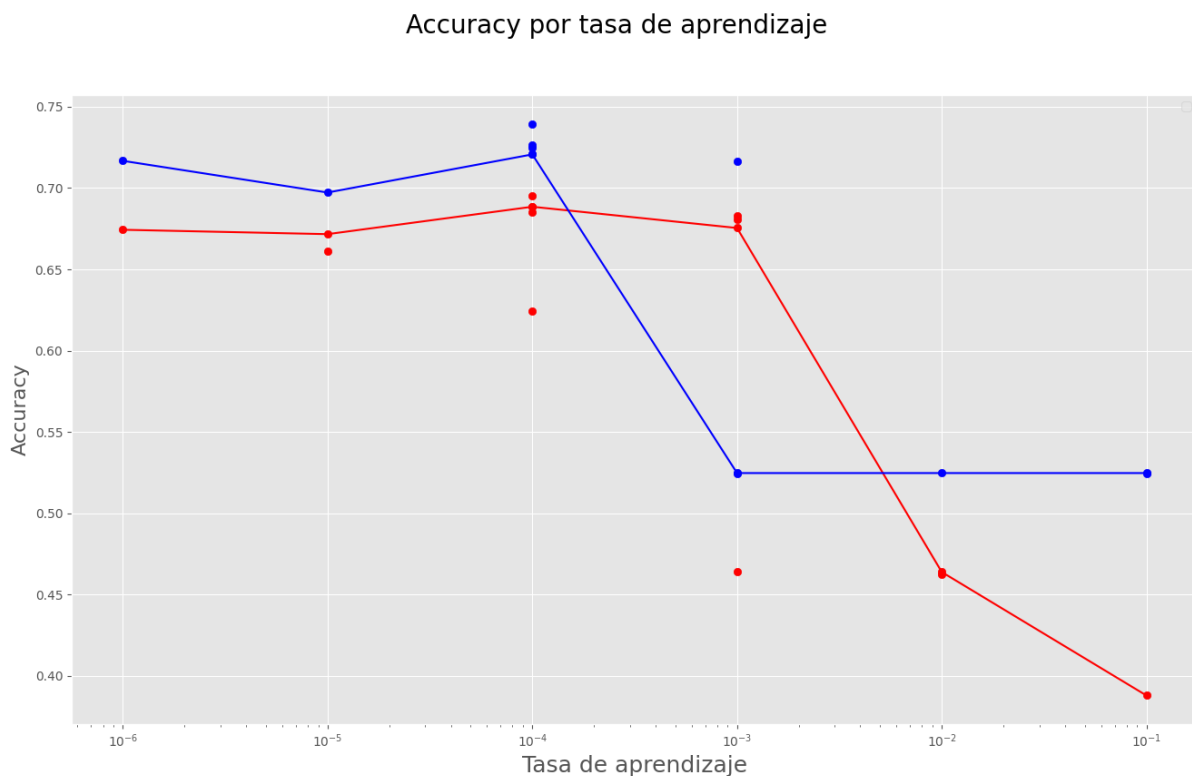


Figura 5-5 Accuracy por tasa de aprendizaje, Red neuronal Multi-layer

5.6 Comparación entre los modelos utilizados para los problemas de clasificación

Comparando los resultados obtenidos en las tres secciones anteriores podemos afirmar que el algoritmo KNN ha funcionado mejor que las redes neuronales en los tres problemas de clasificación. En los *Problemas de clasificación 1* y *2*, KNN ha obtenido mejor *accuracy* casi por una décima. La Red Neuronal Multi-Layer ha funcionado mejor que la Single-Layer en estos dos problemas, pero a penas por unas pocas centésimas. Destaca el *Problema de Clasificación 3*, para el que se han obtenido muy malos resultados con las redes neuronales y sin embargo resultados notoriamente mejores (en cuanto a precisión) para KNN.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Durante el trabajo hemos aplicado distintos algoritmos de aprendizaje automático a varios problemas de regresión y clasificación. Las medidas obtenidas en la mayoría de los casos poseen valores aceptables, pero no excesivamente altos. La conclusión que podemos sacar de este hecho es que el fenómeno de las elecciones políticas existe un componente caótico alto. El algoritmo que mejor ha funcionado para los problemas de clasificación ha sido el Algoritmo KNN, seguido de la Red Neuronal Multi-Layer y por último la Red Neuronal Single-Layer (con resultados muy parecidos a la red neuronal con una capa oculta). Destaca el mal funcionamiento de ambos modelos de red neuronal para el *Problema de clasificación* 3.

6.2 Trabajo futuro

En el futuro podríamos intentar mejorar las predicciones realizadas acerca de las elecciones políticas. Esto se podría conseguir considerando más características poblacionales. También sería positivo que las divisiones territoriales de las cuales se tuviera información sobre las características fueran cada vez menores, ya que, si la información se tuviera a nivel de individuo en lugar de, por ejemplo, municipio, la base de datos representaría mucho mejor la realidad. Para esto se necesitaría que órganos oficiales (como el INE) realizaran más encuestas y estudios, o realizar el estudio por cuenta propia. En el trabajo nos hemos centrado principalmente en las elecciones al gobierno de España, otra consideración que mejoraría la calidad de los resultados sería el hecho de tener en cuenta más países. Por último, otra manera de conseguir resultados más precisos es considerar otros algoritmos de aprendizaje automático y Big data tales como Redes neuronales recurrentes o Redes neuronales convolucionales.

También sería interesante aplicar algoritmos de Aprendizaje automático y Big data a otras cuestiones que tienen que ver con la política, por ejemplo la generación y detección *fake news* (noticias falsas). Aplicaciones “malignas” podrían utilizar los datos recopilados en este trabajo para generar noticias falsas cuyos contenidos sean acordes a la orientación política de una determinada población.

Referencias

- [1] B. Léon, B. Olivier, “The Tradeoffs of Large Scale Learning”, Advances in Neural Information Processing Systems, 2008, pp. 161–168.
- [2] D. Kingma, B. Jimmy, “Adam: A method for stochastic optimization”, arXiv:1412.6980, 2014.
- [3] F. Thomas S., “An inconsistent maximum likelihood estimate”, Journal of the American Statistical Association, 1982, pp. 831–834.
- [4] H. Geoffrey, “Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude”, COURSERA: Neural Networks for Machine Learning, 2020, pp. 26-29.
- [5] H. Rob J., K. Anne B., “Another look at measures of forecast accuracy”, International Journal of Forecasting 22, 2006, pp. 679-688.
- [6] <https://dataverse.harvard.edu/dataverse/eda>, Harvard Dataverse.
- [7] <http://www.infoelectoral.mir.es/infoelectoral/min/areaDescarga.html?method=inicio>, MIR Gobierno de España.
- [8] <http://www.tse.jus.br/eleicoes/estatisticas/estatisticas-eleitorais>, TSE Brasil.
- [9] M. David J. C., “Information Theory, Inference and Learning Algorithms”, Cambridge University Press, 2003, p. 508.
- [10] P. David M. W., “Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation”, Journal of Machine Learning Technologies, 2011, pp. 37-63.
- [11] S. Stephen V., “Selecting and interpreting measures of thematic classification accuracy”, Remote Sensing of Environment, 1997, pp. 77-89.

Anexos

A Normas de estilo utilizadas para la introducción de código en la memoria

En este anexo explicamos las normas de estilo que se han seguido a la hora de insertar código entre el texto de esta memoria. La mayoría del código del trabajo está alojado en Google Colab. Este entorno posee celdas de código y celdas de texto. Para las celdas de texto se utilizará la fuente Courier New con tamaño 12 y color negro. Para las celdas de código se utilizará la fuente Courier New con tamaño 10 y los mismos colores que utiliza el estilo por defecto para Python del programa Notepad ++. Tanto en las celdas de código como en las de texto los bloques estarán alineados a la izquierda. Los programas que no hayan sido escritos en Google Colab sólo poseen código, por lo tanto la fuente utilizada será la segunda de las que acabamos de mencionar. Mostramos a continuación un ejemplo de cómo se vería en esta memoria dos celdas de texto y dos celdas de código intercaladas.

Imports

```
import sys
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import seaborn as sns
from google.colab import drive
from sklearn.metrics import accuracy_score
sns.set()
```

Parámetros e hiperparámetros

```
test_ratio=0.2
dataset = '/content/drive/My
Drive/Archivos_google_colab/Datos/dataset_v1.csv'
n_neighbors = 7 #Numero de vecinos a considerar en KNN
metric = 'euclidean' #Distancia que usar en KNN
```

B Tablas de los cinco partidos principales de Brasil, Estados Unidos y España

Las tablas que se muestran a continuación sólo tienen en cuenta las últimas elecciones a presidente celebradas en cada uno de los países.

Tabla 0-1 Porcentaje de votos a los cinco partidos principales en Brasil

Estado	PT	PSL	PSDB	MDB	PDT
RO	0,0481	0,2342	0,0766	0,1475	0,0272

AM	0,0994	0,0996	0,0974	0,0713	0,1035
CE	0,2312	0,0862	0,0719	0,0703	0,2896
ZZ	0,1010	0,5879	0,0345	0,0049	0,1452
RJ	0,0810	0,2715	0,0178	0,0224	0,0584
SE	0,1809	0,0708	0,0499	0,0652	0,0319
MA	0,1279	0,0844	0,0365	0,1041	0,1568
AL	0,0963	0,1097	0,1410	0,2460	0,0319
PA	0,1697	0,0759	0,0978	0,1923	0,0441
PR	0,0956	0,1474	0,0338	0,0876	0,0344
MG	0,2013	0,1220	0,0816	0,0293	0,0462
RN	0,2215	0,0737	0,0809	0,0637	0,1037
AC	0,2201	0,1525	0,0430	0,1138	0,0356
PB	0,1723	0,0802	0,0875	0,0688	0,0416
PE	0,1955	0,0736	0,0477	0,0685	0,0364
DF	0,0872	0,1155	0,0579	0,0892	0,0428
PI	0,2842	0,0520	0,0440	0,1198	0,0374
RR	0,0350	0,2396	0,0929	0,0777	0,0514
MS	0,1343	0,1966	0,2133	0,1012	0,0927
BA	0,4088	0,0682	0,0434	0,0084	0,0323
TO	0,1310	0,1334	0,0702	0,0343	0,0173
RS	0,1730	0,1670	0,0906	0,1424	0,0650
AP	0,0642	0,1100	0,0456	0,0448	0,1139
SP	0,1397	0,2480	0,1845	0,0560	0,0307
MT	0,0642	0,2277	0,1008	0,0392	0,0213
ES	0,0892	0,1959	0,0626	0,0198	0,0302
SC	0,1118	0,2899	0,0724	0,0970	0,0190
GO	0,0839	0,1361	0,0928	0,0649	0,0300
BRASIL	0,1635	0,1624	0,0917	0,0701	0,0573

Tabla 0-2 Porcentaje de votos a los cinco partidos principales en España

Provincia	PP	PSOE	VOX	Cs	Podemos
Murcia	0,2670	0,2496	0,2816	0,0750	0,0894
Sevilla	0,1810	0,3659	0,1816	0,0804	0,1476
Zaragoza	0,2359	0,3115	0,1817	0,0925	0,1139
Ávila	0,3507	0,2639	0,1867	0,0660	0,0653
Granada	0,2200	0,3349	0,2089	0,0790	0,1241
Salamanca	0,3505	0,2975	0,1808	0,0874	0,0704
Navarra	0,2988	0,2526	0,0586	0,0000	0,1674
Rioja, La	0,3459	0,3523	0,1159	0,0715	0,0997
Palencia	0,3608	0,3360	0,1458	0,0626	0,0817
Jaén	0,2268	0,3918	0,1989	0,0686	0,0994
Balears, Illes	0,2307	0,2571	0,1724	0,0744	0,1829
Barcelona	0,0776	0,2196	0,0635	0,0599	0,1556
Valencia/València	0,2225	0,2725	0,1783	0,0774	0,1390
León	0,2872	0,3381	0,1573	0,0644	0,1057
Pontevedra	0,2934	0,3213	0,0736	0,0463	0,1565
Lugo	0,3849	0,3231	0,0823	0,0326	0,0929
Castellón/Castelló	0,2401	0,2877	0,1877	0,0695	0,1343

Segovia	0,3339	0,3003	0,1734	0,0816	0,0940
Málaga	0,2180	0,3031	0,2168	0,0895	0,1307
Huesca	0,2656	0,3395	0,1536	0,0824	0,1251
Córdoba	0,2278	0,3343	0,1880	0,0825	0,1475
Huelva	0,1999	0,3706	0,2122	0,0743	0,1229
Gipuzkoa	0,0620	0,1820	0,0191	0,0100	0,1506
Toledo	0,2616	0,3223	0,2385	0,0689	0,0966
Bizkaia	0,0888	0,1924	0,0244	0,0111	0,1550
Burgos	0,3118	0,3269	0,1506	0,0826	0,1123
Teruel	0,2376	0,2568	0,1268	0,0506	0,0540
Badajoz	0,2530	0,3882	0,1746	0,0805	0,0927
Melilla	0,2971	0,1654	0,1851	0,0298	0,0263
Araba/Álava	0,1501	0,2206	0,0378	0,0149	0,1659
Almería	0,2604	0,2979	0,2693	0,0762	0,0814
Cádiz	0,1831	0,3102	0,2162	0,0914	0,1541
Lleida	0,0714	0,1457	0,0452	0,0346	0,0794
Cáceres	0,2779	0,3846	0,1618	0,0702	0,0908
Asturias	0,2351	0,3369	0,1606	0,0676	0,1616
Ciudad Real	0,2811	0,3457	0,2105	0,0689	0,0836
Girona	0,0494	0,1492	0,0523	0,0392	0,0954
Palmas, Las	0,2154	0,2939	0,1344	0,0584	0,1555
Coruña, A	0,3087	0,3038	0,0830	0,0470	0,1271
Albacete	0,2769	0,3287	0,2085	0,0751	0,0978
Alicante/Alacant	0,2454	0,2843	0,1981	0,0818	0,1285
Madrid	0,2511	0,2709	0,1849	0,0914	0,1312
Zamora	0,3398	0,3322	0,1728	0,0693	0,0710
Santa Cruz de Tenerife	0,2030	0,2880	0,1162	0,0499	0,1404
Ourense	0,3980	0,3353	0,0782	0,0377	0,0799
Cantabria	0,2599	0,2336	0,1500	0,0480	0,0872
Cuenca	0,3119	0,3762	0,1860	0,0471	0,0696
Ceuta	0,2249	0,3161	0,3554	0,0344	0,0393
Tarragona	0,0782	0,1924	0,0813	0,0596	0,1219
Valladolid	0,2979	0,3061	0,1828	0,0874	0,1113
Guadalajara	0,2324	0,3142	0,2418	0,0776	0,1137
Soria	0,3337	0,3511	0,1371	0,0584	0,0770
ESPANNA	0,2141	0,2825	0,1521	0,0686	0,1298

Tabla 0-3 Porcentaje de votos a los cinco partidos principales en Estados Unidos

Estado	DEM	REP	LIB	GREEN	CON
New York	0,5614	0,3239	0,0074	0,0138	0,0375
USA	0,4801	0,4583	0,0302	0,0100	0,0021
Missouri	0,3814	0,5677	0,0347	0,0091	0,0000
Iowa	0,4175	0,5116	0,0378	0,0073	0,0000
Montana	0,3594	0,5647	0,0567	0,0161	0,0000
Oregon	0,5007	0,3909	0,0471	0,0250	0,0000
New Mexico	0,4826	0,4004	0,0934	0,0124	0,0000
Mississippi	0,4011	0,5794	0,0119	0,0031	0,0000

Illinois	0,5583	0,3876	0,0379	0,0139	0,0000
South Carolina	0,4067	0,5494	0,0234	0,0062	0,0000
Vermont	0,5572	0,2976	0,0314	0,0211	0,0000
Rhode Island	0,5441	0,3890	0,0318	0,0134	0,0000
Kentucky	0,3268	0,6252	0,0279	0,0072	0,0000
Indiana	0,3777	0,5694	0,0490	0,0029	0,0000
California	0,6173	0,3162	0,0337	0,0196	0,0000
Hawaii	0,6098	0,2944	0,0365	0,0291	0,0000
Florida	0,4782	0,4902	0,0220	0,0068	0,0000
New Jersey	0,5545	0,4135	0,0187	0,0098	0,0000
Virginia	0,4975	0,4443	0,0297	0,0069	0,0000
Maine	0,4635	0,4348	0,0494	0,0185	0,0000
District of Columbia	0,9048	0,0407	0,0157	0,0136	0,0000
Alabama	0,3436	0,6208	0,0209	0,0044	0,0000
Massachusetts	0,5905	0,3229	0,0408	0,0141	0,0000
Pennsylvania	0,4785	0,4858	0,0240	0,0082	0,0000
Delaware	0,5335	0,4192	0,0334	0,0138	0,0000
Louisiana	0,3845	0,5809	0,0187	0,0069	0,0000
Colorado	0,4816	0,4325	0,0518	0,0138	0,0000
Alaska	0,3655	0,5128	0,0588	0,0180	0,0000
Minnesota	0,4645	0,4493	0,0384	0,0126	0,0000
Texas	0,4324	0,5223	0,0316	0,0080	0,0000
Wisconsin	0,4645	0,4722	0,0358	0,0104	0,0000
West Virginia	0,2648	0,6863	0,0323	0,0113	0,0000
Arizona	0,4513	0,4867	0,0413	0,0133	0,0000
New Hampshire	0,4683	0,4646	0,0414	0,0087	0,0000
Connecticut	0,5457	0,4093	0,0296	0,0139	0,0000
Arkansas	0,3365	0,6057	0,0264	0,0084	0,0000
Maryland	0,6033	0,3392	0,0286	0,0129	0,0000
North Dakota	0,2723	0,6296	0,0622	0,0110	0,0000
Washington	0,5254	0,3683	0,0485	0,0176	0,0000
Michigan	0,4727	0,4750	0,0359	0,0107	0,0000
Ohio	0,4356	0,5169	0,0000	0,0084	0,0000
Oklahoma	0,2893	0,6532	0,0575	0,0000	0,0000
Nebraska	0,3370	0,5875	0,0461	0,0000	0,0000
Utah	0,2746	0,4554	0,0350	0,0000	0,0000
Idaho	0,2749	0,5926	0,0410	0,0000	0,0000
Nevada	0,4792	0,4550	0,0332	0,0000	0,0000
Georgia	0,4564	0,5077	0,0305	0,0000	0,0000
Kansas	0,3605	0,5665	0,0468	0,0000	0,0000
South Dakota	0,3174	0,6153	0,0563	0,0000	0,0000
North Carolina	0,4617	0,4983	0,0274	0,0000	0,0000
Wyoming	0,2163	0,6740	0,0513	0,0000	0,0000
Tennessee	0,3472	0,6072	0,0000	0,0000	0,0000

Identificando cada unidad territorial con un vector de cinco dimensiones cuyos componentes son los porcentajes de votos a los cinco partidos principales de cada país se han elaborado tablas en la que se muestran las distancias de cada unidad territorial al vector país, ordenadas de menos a mayor.

Tabla 0-4 Distancias políticas Brasil

Unidad territorial	Distancia
BRASIL	0,0000
MG	0,0049
TO	0,0052
RS	0,0054
PB	0,0071
GO	0,0078
AC	0,0081
PR	0,0090
DF	0,0097
AM	0,0102
ES	0,0108
SE	0,0111
PE	0,0113
RN	0,0135
MT	0,0165
SP	0,0174
AP	0,0186
MS	0,0190
MA	0,0214
SC	0,0215
RR	0,0226
PA	0,0227
RO	0,0256
RJ	0,0264
PI	0,0319
AL	0,0413
CE	0,0648
BA	0,0758
ZZ	0,2002

Tabla 0-5 Distancias políticas España

Unidad territorial	Distancia
ESPANNA	0,0000
Valencia/València	0,0010
Palmas, Las	0,0012
Santa Cruz de Tenerife	0,0019
Castellón/Castelló	0,0020
Zaragoza	0,0030
Madrid	0,0031

Alicante/Alacant	0,0033
Balears, Illes	0,0042
Asturias	0,0045
Córdoba	0,0047
Málaga	0,0051
Huesca	0,0061
Granada	0,0062
Cantabria	0,0067
Cádiz	0,0070
Teruel	0,0079
León	0,0091
Valladolid	0,0092
Sevilla	0,0094
Guadalajara	0,0097
Albacete	0,0103
Huelva	0,0116
Burgos	0,0120
Toledo	0,0124
Ciudad Real	0,0140
Coruña, A	0,0147
Badajoz	0,0147
Pontevedra	0,0152
Jaén	0,0152
Cáceres	0,0161
Segovia	0,0166
Almería	0,0185
Soria	0,0221
Zamora	0,0221
Murcia	0,0223
Navarra	0,0230
Salamanca	0,0235
Cuenca	0,0236
Ávila	0,0244
Rioja, La	0,0245
Araba/Álava	0,0252
Palencia	0,0268
Barcelona	0,0312
Tarragona	0,0317
Melilla	0,0339
Lugo	0,0384
Bizkaia	0,0441
Ourense	0,0455
Ceuta	0,0519
Lleida	0,0542

Gipuzkoa	0,0548
Girona	0,0569

Tabla 0-6 Distancias políticas Estados Unidos

Unidad territorial	Distancia
USA	0,0000
Nevada	0,0001
New Hampshire	0,0003
Michigan	0,0004
Minnesota	0,0004
Wisconsin	0,0005
Virginia	0,0005
Pennsylvania	0,0008
Florida	0,0011
Colorado	0,0012
Maine	0,0013
Arizona	0,0018
North Carolina	0,0020
Georgia	0,0031
Delaware	0,0044
Oregon	0,0055
Ohio	0,0063
Texas	0,0064
Connecticut	0,0067
Iowa	0,0068
New Mexico	0,0074
New Jersey	0,0077
Rhode Island	0,0089
Washington	0,0105
Illinois	0,0112
South Carolina	0,0137
Alaska	0,0170
Mississippi	0,0213
Missouri	0,0218
Indiana	0,0232
Louisiana	0,0243
Kansas	0,0264
New York	0,0264
Montana	0,0266
Maryland	0,0294
Massachusetts	0,0307
Vermont	0,0319
Nebraska	0,0375

California	0,0391
Tennessee	0,0409
Arkansas	0,0424
Utah	0,0424
Hawaii	0,0441
Alabama	0,0452
Kentucky	0,0514
South Dakota	0,0519
Idaho	0,0604
North Dakota	0,0736
Oklahoma	0,0753
West Virginia	0,0984
Wyoming	0,1167
District of Columbia	0,3550

C Identificadores numéricos características

A continuación adjuntamos una tabla que asigna a cada característica poblacional considerada en la base de datos Elecciones al presidente de gobierno en España un identificador numérico.

Tabla 0-7 Identificadores características poblacionales

Característica	Identificador
Religiosidad	1
Edad media	2
Escolarización	3
Esperanza de vida	4
Gobierno de la comunidad	5
Gobierno de España	6
Gobierno de Estados Unidos	7
Habitantes municipio	8
IDH	9
Acceso a internet	10
PIB per cápita	11
Criminalidad	12
Inmigración de países en vías de desarrollo	13
Inmigración europea	14
Viajeros	15
Porcentaje de mujeres	16
Latitud	17

Longitud	18
Altitud	19
Candidatura	20

D Script Python para la obtención del número de habitantes de cada municipio

```

from time import sleep
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

#Abre el navegador
navegador = webdriver.Firefox(executable_path='/usr/bin/geckodriver')

hechos=0

with open('habitantes_amao.csv', 'r',encoding="utf-8") as f:
    for linea in f:
        municipio = linea.replace('\n','')
        #Poner lo que hay entre parentesis delante
        if (municipio.find("(")!=-1):
            start = municipio.find("(") + len("(")
            end = municipio.find(")")
            busqueda = municipio[start:end] + ' ' +
municipio[0:start-1]
        else:
            busqueda=municipio

        url_empezar='https://www.google.com/search?q=habitantes'+busqueda+
'+municipio'
        navegador.get(url_empezar)
        dato_xp =
'/html/body/div[6]/div[2]/div[9]/div[1]/div[2]/div/div[2]/div[2]/div/div/
div[1]/div[1]/div[1]/div[1]/div/div[2]/div/div[1]/div/div/div[1]'
        try:
            valor = navegador.find_element_by_xpath(dato_xp).text
        except:
            dato_xp =
'/html/body/div[6]/div[2]/div[9]/div[1]/div[2]/div/div[2]/div[2]/div/div/
div[1]/div[1]/div[1]/div[1]/div/div[2]/div/div/div[1]/div'
            try:
                valor = navegador.find_element_by_xpath(dato_xp).text
            except:
                dato_xp =
'/html/body/div[6]/div[2]/div[9]/div[1]/div[2]/div/div[2]/div[2]/div/div/
div[1]/div[1]/div/div[1]/div/div[2]/div/div[1]'
                try:
                    valor = navegador.find_element_by_xpath(dato_xp).text
                except:
                    dato_xp =
'/html/body/div[6]/div[2]/div[9]/div[1]/div[2]/div/div[2]/div[2]/div/div/
div[1]/div[1]/div/div[1]/div/div[2]/div/span/span'
                    try:
                        valor = navegador.find_element_by_xpath(dato_xp).text
                    except:
                        break
        print(municipio+'\t'+valor)

```


E Clasificación de los partidos políticos

Tabla 0-8 Clasificación de los partidos políticos

Partido político	Categoría
ECP	Izquierda
ECP-GUANYEM	Izquierda
EN COMÚ	Izquierda
EN MAREA	Izquierda
IC-V	Izquierda
IR-ERG	Izquierda
IR-PRE	Izquierda
I.U.	Izquierda
IU	Izquierda
IU-BA	Izquierda
IU-CHA-UPeC	Izquierda
IU-CM	Izquierda
IUCM-LV	Izquierda
IUCyL	Izquierda
IU-IX	Izquierda
IULV-CA	Izquierda
IULV-CA, UP	Izquierda
IULV-CA,UPe	Izquierda
IU-UPeC	Izquierda
IUV-RM	Izquierda
LV	Izquierda
LV-GV	Izquierda
MÉS	Izquierda
MÉS COMPROM	Izquierda
MÉS-ESQUERR	Izquierda
PACMA	Izquierda
PCOE	Izquierda
PCPA	Izquierda
PCPC	Izquierda
P.C.P.E.	Izquierda
PCPE	Izquierda
PCTE	Izquierda
PH	Izquierda
PODEMOS	Izquierda
PODEMOS-AHA	Izquierda
PODEMOS/AHAL DUGU-IU-EQUO	Izquierda

PODEMOS-Aho	Izquierda
PODEMOS - C	Izquierda
PODEMOS-COM	Izquierda
PODEMOS-En	Izquierda
PODEMOS-EN	Izquierda
PODEMOS-EN MAREA-ANOVA-EU	Izquierda
PODEMOS-EU	Izquierda
PODEMOS-EU-	Izquierda
PODEMOS-EUI	Izquierda
PODEMOS-EUIB	Izquierda
PODEMOS-EUP	Izquierda
PODEMOS-EUPV	Izquierda
PODEMOS-IU	Izquierda
PODEMOS-IU-	Izquierda
PODEMOS-IU-BATZARRE	Izquierda
PODEMOS-IU-EQUO	Izquierda
PODEMOS-IU-EQUO-BATZARRE	Izquierda
PODEMOS-IU-EQUO-CLIAS	Izquierda
PODEMOS-IU-EQUO-IZCA	Izquierda
PODEMOS-IU-EQUO-SEGOVIEMOS	Izquierda
PODEMOS-IU LV CA	Izquierda
PODEMOS-IX	Izquierda
PODEMOS-IX-	Izquierda
RECORTES CE	Izquierda
RECORTES CERO-GRUPO VERDE	Izquierda
RECORTES CERO-GV	Izquierda
SAIn	Izquierda
UP: IULV-CA	Izquierda
UP-UPeC	Izquierda
AVE	Izquierda
IUCLM-LV	Izquierda
AuN	Derecha
ES2000	Derecha
DN	Derecha
FA	Derecha
FE	Derecha
FE de las J	Derecha
FE de las JONS	Derecha
FEI-FE 2000	Derecha
GIL	Derecha
NA+	Derecha
PADE	Derecha
PFyV	Derecha
P.P.	Derecha

PP	Derecha
P.P-E.U.	Derecha
PP-EU	Derecha
PP-FORO	Derecha
PP-PAR	Derecha
PP-UPM	Derecha
UPN-PP	Derecha
VOX	Derecha
UN	Derecha
EB	Centro-izquierda
PSC	Centro-izquierda
PSC (PSC-PSOE)	Centro-izquierda
PSC-PSOE	Centro-izquierda
PSA	Centro-izquierda
PSD	Centro-izquierda
PSdeG-PSOE	Centro-izquierda
PSDEG-PSOE	Centro-izquierda
PSE-EE	Centro-izquierda
PSE-EE (PSO	Centro-izquierda
PSE-EE (PSOE)	Centro-izquierda
PSE-EE-PSOE	Centro-izquierda
PSE-EE(PSOE)	Centro-izquierda
PSM-IV-EXM-EQUO	Centro-izquierda
P.S.O.E.	Centro-izquierda
PSOE	Centro-izquierda
PSOE-A	Centro-izquierda
PSOE de A	Centro-izquierda
PSOE-NCa	Centro-izquierda
PSOE-PROGR.	Centro-izquierda
POSI	Centro-izquierda
PUM+J	Centro-izquierda
CDS	Centro-derecha
C's	Centro-derecha
Cs	Centro-derecha
Cs	Centro-derecha
P-LIB	Centro-derecha
UPL	Centro-derecha
UPyD	Centro-derecha
UPYD	Centro-derecha
AMAIUR	Regionalista-izquierda
aralar	Regionalista-izquierda
ARALAR-ZUTIK	Regionalista-izquierda
AxSí	Regionalista-izquierda
B.N.G.	Regionalista-izquierda

BNG	Regionalista-izquierda
BNG-NÓS	Regionalista-izquierda
CHA	Regionalista-izquierda
CHA-IU	Regionalista-izquierda
COMPROMÍS 2	Regionalista-izquierda
CpM	Regionalista-izquierda
CUP-PR	Regionalista-izquierda
DL	Regionalista-izquierda
EA	Regionalista-izquierda
EH Bildu	Regionalista-izquierda
EKA	Regionalista-izquierda
ERC	Regionalista-izquierda
ERC-CATSI	Regionalista-izquierda
ERC-CATSÍ	Regionalista-izquierda
ERC-RI.cat	Regionalista-izquierda
ERC - RI.cat / ESQUERRA	Regionalista-izquierda
ERC-SOBIRAN	Regionalista-izquierda
ERC-SOBIRANISTES	Regionalista-izquierda
ERPv	Regionalista-izquierda
esquerra	Regionalista-izquierda
ESQUERRA	Regionalista-izquierda
GBAI	Regionalista-izquierda
Na-Bai	Regionalista-izquierda
NA-BAI	Regionalista-izquierda
NCa	Regionalista-izquierda
NC-CCa-PNC	Regionalista-izquierda
NC-CCN	Regionalista-izquierda
PA	Regionalista-izquierda
PARTIDO RIOJANO	Regionalista-izquierda
PRAO	Regionalista-izquierda
P.R.C.	Regionalista-izquierda
PRC	Regionalista-izquierda
PYLN	Regionalista-izquierda
ARA-MES-ESQ	Regionalista-izquierda
FPG	Regionalista-izquierda
NÓS	Regionalista-izquierda
AUNACV	Regionalista-derecha
BLM	Regionalista-derecha
CC	Regionalista-derecha
CCa-PNC	Regionalista-derecha
CCa-PNC-NC	Regionalista-derecha
CC-NC-PNC	Regionalista-derecha
CC-PNC	Regionalista-derecha
CC-PNC-PIL	Regionalista-derecha

CDC	Regionalista-derecha
CDN	Regionalista-derecha
CiU	Regionalista-derecha
CIU	Regionalista-derecha
E.A.J.-P.N.V.	Regionalista-derecha
EAJ-PNV	Regionalista-derecha
EL PI	Regionalista-derecha
FAC	Regionalista-derecha
JxCAT-JUNTS	Regionalista-derecha
PAR	Regionalista-derecha
PIL	Regionalista-derecha
PPSO	Regionalista-derecha
PREPAL	Regionalista-derecha
iTERUEL EXI	Regionalista-derecha
UM	Regionalista-derecha
UM-INME	Regionalista-derecha
unio.cat	Regionalista-derecha
UV	Regionalista-derecha
XAV	Regionalista-derecha
CCS	Regionalista-derecha

Tabla 0-9 Clasificación partidos Brasil y Estados Unidos

Partido político	Categoría
Partido dos Trabalhadores	Izquierda
Partido Social Liberal	Derecha
Partido da Social Democracia Brasileira	Derecha
Movimento Democrático Brasileiro	Izquierda
Partido Democrático Trabalhista	Izquierda
democrat	Izquierda
republican	Derecha
libertarian	Derecha
green	Izquierda
conservative	Derecha

F Función carga_datos_csv

Las librerías utilizadas para implementar los algoritmos de aprendizaje automático necesitan consumir los datos en un formato específico. Este formato no es el mismo que obtenemos al exportar nuestra tabla SQL. Debido a esto, se ha escrito una función cuyo objetivo es la transformación de estos datos al formato correcto, además de solucionar el problema de las variables categóricas mediante One Hot Encoding y de la normalización de los datos. También esta función divide el conjunto de datos en un conjunto de entrenamiento y un conjunto de test. Adjuntamos y comentamos a continuación la entrada y salida de esta función, que ha sido importante en el desarrollo del trabajo.

```
#Función para cargar los datos normalizados y procesados mediante hot
encoding en arrays numpy divididos en conjunto de test y conjunto de
entrenamiento
#Devuelve las listas numpy: (x_columns,y_columns), (x_train, y_train),
(x_test, y_test)
#porcentaje_test: indica el porcentaje de los datos de muestra que se
utilizará para el conjunto de test
#data: DataFrame de la librería pandas con el dataset
#categoría: identifica la columna que es la categoría
#hot_encoded: lista con los nombres de las columnas que van a ser
procesadas por hot encoding
#normalizados: lista con los nombres de las columnas que van a ser
normalizadas
#seed: semilla para los procesos aleatorios que ocurren dentro de la
función
def
carga_datos_csv(data,categoría,porcentaje_test=0.25,hot_encoded=None,norm
alizados=None,seed=None):
    #Normalización
    for col in normalizados:
        data[col]=(data[col]-data[col].min())/(data[col].max()-
data[col].min())
    #Divide entre test y train
    train, test = train_test_split(data,
test_size=porcentaje_test,random_state=seed)
    #Separar el label
    label_train = pd.DataFrame(train[categoría])
    train.drop([categoría],axis=1, inplace=True)
    label_test = pd.DataFrame(test[categoría])
    test.drop([categoría],axis=1, inplace=True)
    #Hot encoding
    for col in hot_encoded:
        if col==categoría:
            label_train =
pd.concat([label_train,pd.get_dummies(label_train[col],
prefix=col)],axis=1)
            label_train.drop([col],axis=1, inplace=True)
            label_test =
pd.concat([label_test,pd.get_dummies(label_test[col],
prefix=col)],axis=1)
            label_test.drop([col],axis=1, inplace=True)
        else:
            train = pd.concat([train,pd.get_dummies(train[col],
prefix=col)],axis=1)
            train.drop([col],axis=1, inplace=True)
            test = pd.concat([test,pd.get_dummies(test[col],
prefix=col)],axis=1)
            test.drop([col],axis=1, inplace=True)
```

```

    #Rellenar el conjunto de entrenamiento con las columnas que estan en
    el conjunto de test pero no en el de entrenamiento y viceversa
    (características)
    missing_cols = set( train.columns ) - set( test.columns )
    for c in missing_cols:
        test[c] = 0
    test = test[train.columns]
    missing_cols = set( test.columns ) - set( train.columns )
    for c in missing_cols:
        train[c] = 0
    train = train[test.columns]
    #Rellenar el conjunto de test con las columnas que estan en el
    conjunto de entrenamiento pero no en el de test y viceversa (categorías)
    missing_cols = set( label_train.columns ) - set( label_test.columns )
    for c in missing_cols:
        label_test[c] = 0
    label_test = label_test[label_train.columns]
    missing_cols = set( label_test.columns ) - set( label_train.columns )
    for c in missing_cols:
        label_train[c] = 0
    label_train = label_train[label_test.columns]

    #Obtener las listas ordenadas de las columnas de las características
    y las categorías
    x_columns= train.columns
    y_columns= label_train.columns

    #Convertir a array de numpy
    x_train = train.to_numpy()
    y_train = label_train.to_numpy()
    x_test = test.to_numpy()
    y_test = label_test.to_numpy()

    return (x_columns,y_columns), (x_train, y_train), (x_test, y_test)

```

La función `carga_datos_csv` recibe los siguientes parámetros:

- **Data.** DataFrame bidimensional con ejes etiquetados que contiene el conjunto de los datos que se desea procesar.
- **Categoría.** Etiqueta de la “columna” que se va a usar como categoría, en base a la cual se desea clasificar el resto de datos.
- **Porcentaje_test.** Porcentaje de los datos proporcionados que se destinarán al test, no al entrenamiento.
- **Hot_encoded.** Lista conteniendo las etiquetas de las columnas a las cuales se desea realizar el proceso de One Hot Encoding. Las características categóricas han de ser transformadas a un formato numérico para poder ser manejadas por los algoritmos. Cuando no existe una relación de orden, una manera de realizar esto es mediante el proceso llamado One Hot Encoding. Explicaremos este proceso mediante un ejemplo. Imaginemos que nuestra variable categórica es “Color de ojos” y que puede tomar los valores “Verdes”, “Azules” y “Marrones”. La

columna conteniendo los datos de “Color de ojos” se convierte en tres columnas “Ojos verdes”, “Ojos azules” y “Ojos marrones”. Una celda cuyo valor antes fuera “Verdes” es convertida a los valores [1,0,0] de estas nuevas columnas.

- **Normalizados.** Lista conteniendo las etiquetas de las columnas a las cuales se desea normalizar. Se realiza la normalización dividiendo el valor por la diferencia entre el máximo y el mínimo valor de la característica.
- **Seed.** Semilla del proceso que ordena los datos introducidos aleatoriamente para realizar después la separación entre conjunto de test y conjunto de entrenamiento.

La función devuelve tres pares de listas numpy. El primer par (x_columns, y_columns) contiene una lista de las etiquetas ordenadas del mismo modo que los pares (x_train, y_train) y (x_test, y_test), ya que en el interior de esta función el orden puede haber variado respecto al orden que poseía el DataFrame de entrada. El segundo par (x_train, y_train) contiene los datos que se usarán para la fase de entrenamiento. El tercer par (x_test, y_test) contiene los datos que se usarán para la fase de test. Esta función realiza una llamada internamente a la función *train_test_split* de Sklearn, que es donde reside la mayor parte de la funcionalidad.

G Matrices de covarianzas, varianzas, medias, máximos y mínimos

En la sección Estudio estadístico básico hemos mostrado las variables con más correlación para el *Problema de regresión 1*, a continuación mostramos las mismas tablas para los *Problemas de regresión 2, 3, 4*.

Tabla 0-10 Variables con más correlación Problema de regresión 2

Variable	Más correlación	Valor	Menos correlación	Valor
Izquierda	gobierno_usa_rep	0,2745	gobierno_comunidad_c_der	-0,0028
Derecha	gobierno_usa_dem	0,2745	provincia_huesca	-0,0025
Variable	No categórica más correlación	Valor	No categórica menos correlación	Valor
Izquierda	anno	0,1564	edad_media	-0,0142
Derecha	criminalidad	0,1937	viajeros	-0,0190

Tabla 0-11 Variables con más correlación Problema de regresión 3

Variable	Más correlación	Valor	Menos correlación	Valor
----------	-----------------	-------	-------------------	-------

No regionalista	gobierno_comunidad_region_der	-0,5114	mujeres	0,0000
Regionalista	gobierno_comunidad_region_der	0,5114	mujeres	0,0000
Variable	No categórica más correlación	Valor	No categórica menos correlación	Valor
No regionalista	religiosidad	0,4257	mujeres	0,0000
Regionalista	religiosidad	-0,4257	mujeres	0,0000

En la sección Estudio estadístico básico hemos mostrado la media, varianza máximos y mínimos de cada variable dependiente para el *Problema de regresión 1*, a continuación mostramos estos datos para el *Problema de regresión 4*. Mostramos los datos de Brasil, Estados Unidos, España y el conjunto de los tres países. En la Tabla 0-12, Tabla 0-13, Tabla 0-14 y Tabla 0-15 mostramos la varianza, la media, el máximo y el mínimo de las variables dependientes del *Problema de regresión 4*. La Tabla 0-12 considera Brasil, Estados Unidos y España a la vez, la Tabla 0-13 sólo considera Estados Unidos, la tabla Tabla 0-14 considera sólo España, la tabla Tabla 0-15 considera sólo Brasil.

Tabla 0-12 Media, varianza, máximos y mínimos Brasil, Estados Unidos y España

Variable	Media	Máximo	Lugar máximo	Mínimo	Lugar mínimo	Varianza
Izquierda	0,3828	0,9185	District of Columbia	0,0753	RO	0,0188
Derecha	0,4338	0,7253	Wyoming	0,0564	District of Columbia	0,0272

Tabla 0-13 Media, varianza, máximos y mínimos Estados Unidos

Variable	Media	Máximo	Lugar máximo	Mínimo	Lugar mínimo	Varianza
Izquierda	0,4558	0,9185	District of Columbia	0,2163	Wyoming	0,0152
Derecha	0,5183	0,7253	Wyoming	0,0564	District of Columbia	0,0148

Tabla 0-14 Media, varianza, máximos y mínimos España

Variable	Media	Máximo	Lugar máximo	Mínimo	Lugar mínimo	Varianza
Izquierda	0,4073	0,5134	Sevilla	0,1918	Melilla	0,0045
Derecha	0,4655	0,6236	Murcia	0,0911	Gipuzkoa	0,0181

Tabla 0-15 Media, varianza, máximos y mínimos Brasil

Variable	Media	Máximo	Lugar máximo	Mínimo	Lugar mínimo	Varianza
Izquierda	0,2074	0,5209	CE	0,0753	RO	0,0104
Derecha	0,2243	0,4325	SP	0,0960	PI	0,0080

Se ha calculado la matriz de covarianzas para cada problema de regresión. Las mostramos a continuación. Un color más cercano al rojo puro indica un valor más cercano a 1. Un color

más cercano al azul puro indica un valor más cercano a -1. Valores más cercanos a 0 toman colores más oscuros.

Tabla 0-16 Matriz de covarianzas Estados Unidos Problema de regresión 4

	IZQ	DER	IDH
IZQ	1	-0,95697389	0,3644281
DER	-0,95697389	1	-0,43009866
IDH	0,3644281	-0,43009866	1

Tabla 0-17 Matriz de covarianzas Brasil Problema de regresión 4

	IZQ	DER	IDH
IZQ	1	0,52327821	0,37542872
DER	0,52327821	1	0,54345227
IDH	0,37542872	0,54345227	1

Tabla 0-18 Matriz de covarianzas España Problema de regresión 4

	IZQ	DER	IDH
IZQ	1	0,38185288	0,08214604
DER	0,38185288	1	0,33589563
IDH	0,08214604	0,33589563	1

Tabla 0-19 Matriz de covarianzas Brasil, Estados Unidos y España Problema de regresión 4

	IZQ	DER	IDH
IZQ	1	0,24700841	0,60571091
DER	0,24700841	1	0,52417329
IDH	0,60571091	0,52417329	1

Las matrices de covarianzas de los *Problemas de regresión 1, 2 y 3* poseen demasiadas variables como para mostrar la tabla completa (a las variables categóricas se las ha aplicado One Hot Encoding, lo que hace que el número de variables crezca mucho). Adjuntamos estas matrices de covarianzas conservando solo algunas de las variables. Para el *Problema de regresión 1* el orden de filas y columnas de la matriz es anno, religiosidad, edad_media, escolarizacion, esperanza_vida, habitantes_municipio, idh, internet, pibpercapita, criminalidad, inmigracion_desarrollo, inmigracion_eur, viajeros, mujeres, candidatura_c_der, candidatura_c_izq, candidatura_der, candidatura_izq, candidatura_region_der, candidatura_region_izq. Para el *Problema de regresión 2* el orden

de filas y columnas de la matriz es anno, religiosidad, edad_media, escolarizacion, esperanza_vida, habitantes_municipio, idh, internet, pibpercapita, criminalidad, inmigracion_desarrollo, inmigracion_eur, viajeros, mujeres, latitud, altitud, longitud, candidatura_der, candidatura_izq Para el *Problema de regresión 3* el orden de filas y columnas de la matriz es anno, religiosidad, edad_media, escolarizacion, esperanza_vida, habitantes_municipio, idh, internet, pibpercapita, criminalidad, inmigracion_desarrollo, inmigracion_eur, viajeros, mujeres, latitud, altitud, longitud, no_regionalista, regionalista.

Tabla 0-20 Matriz de covarianzas Problema de regresión 1

1	0	0,5	0,2	0,8	0	0	0	0,4	-0,3	0,3	-0,2	0,1	-0	0,1	0	-0,1	0,1	-0,1	0,2
0	1	-0,1	-0,3	-0	0	-0,2	-0,3	-0,4	0,1	-0,3	0,1	-0,2	-0,1	0	0	0,3	-0,2	-0,3	-0,3
0,5	-0,1	1	0,2	0,6	-0,1	0,3	-0,3	0,2	-0,1	-0,2	-0	-0,1	0	0	-0,1	0	0,1	0	0,1
0,2	-0,3	0,2	1	0,5	0,4	0,8	0,6	0,8	-0,3	0,3	-0,1	-0,1	0,2	0,1	-0	-0,1	0,1	0,2	0,2
0,8	-0	0,6	0,5	1	0,2	0,5	0,1	0,7	-0,2	0,4	-0,1	0,1	0	0,1	-0,1	-0	0,2	-0	0,2
0	0	-0,1	0,4	0,2	1	0,3	0,4	0,4	-0,2	0,2	-0,1	-0,1	0,2	0	-0	0	0	-0	-0
0	-0,2	0,3	0,8	0,5	0,3	1	0,4	0,7	-0,1	0,2	-0,2	-0,1	0,1	0	-0,1	-0	0,1	0,2	0,1
0	-0,3	-0,3	0,6	0,1	0,4	0,4	1	0,5	-0,2	0,3	-0,1	-0	0,2	0	0	-0,2	0,1	0,1	0
0,4	-0,4	0,2	0,8	0,7	0,4	0,7	0,5	1	-0,2	0,6	-0,1	0,1	0,1	0,1	-0	-0,2	0,2	0,1	0,2
-0,3	0,1	-0,1	-0,3	-0,2	-0,2	-0,1	-0,2	-0,2	1	-0,1	0,2	0,2	-0,1	-0	-0,2	0,2	-0	-0	-0,1
0,3	-0,3	-0,2	0,3	0,4	0,2	0,2	0,3	0,6	-0,1	1	-0	0,2	0	0	-0,1	-0,1	0,1	0,1	0,2
-0,2	0,1	-0	-0,1	-0,1	-0,1	-0,2	-0,1	-0,1	0,2	-0	1	-0	-0,1	-0	-0	0,1	-0	-0	-0
0,1	-0,2	-0,1	-0,1	0,1	-0,1	-0,1	-0	0,1	0,2	0,2	-0	1	-0,1	0	-0	-0	0	-0	0
-0	-0,1	0	0,2	0	0,2	0,1	0,2	0,1	-0,1	0	-0,1	-0,1	1	0	0	-0,1	0	0	-0
0,1	0	0	0,1	0,1	0	0	0	0,1	-0	0	-0	0	0	1	-0,1	-0,1	-0	-0	-0
0	0	-0,1	-0	-0,1	-0	-0,1	0	-0	-0,2	-0,1	-0	-0	0	-0,1	1	-0,7	-0,2	-0,2	-0,2
-0,1	0,3	0	-0,1	-0	0	-0	-0,2	-0,2	0,2	-0,1	0,1	-0	-0,1	-0,1	-0,7	1	-0,2	-0,2	-0,2
0,1	-0,2	0,1	0,1	0,2	0	0,1	0,1	0,2	-0	0,1	-0	0	0	-0	-0,2	-0,2	1	-0,1	-0
-0,1	-0,3	0	0,2	-0	-0	0,2	0,1	0,1	-0	0,1	-0	-0	0	-0	-0,2	-0,2	-0,1	1	-0
0,2	-0,3	0,1	0,2	0,2	-0	0,1	0	0,2	-0,1	0,2	-0	0	-0	-0	-0,2	-0,2	-0	-0	1

Tabla 0-21 Matriz de covarianzas Problema de regresión 2

1	0	0,5	0,2	0,8	0	0	0	0,4	-0	0,3	-0	0,1	-0	-0	0	-0	-0	0,2
0	1	-0	-0	-0	0	-0	-0	-0	0,1	-0	0,1	-0	-0	-0	0,5	-0	0,2	-0
0,5	-0	1	0,2	0,6	-0	0,3	-0	0,2	-0	-0	-0	-0	0	0,5	0,2	-0	0	-0
0,2	-0	0,2	1	0,5	0,4	0,8	0,6	0,8	-0	0,3	-0	-0	0,2	0,4	0,1	0,2	-0	0
0,8	-0	0,6	0,5	1	0,2	0,5	0,1	0,7	-0	0,4	-0	0,1	0	0,3	0,3	0,2	-0	0
0	0	-0	0,4	0,2	1	0,3	0,4	0,4	-0	0,2	-0	-0	0,2	0	0,1	0,1	0	-0
0	-0	0,3	0,8	0,5	0,3	1	0,4	0,7	-0	0,2	-0	-0	0,1	0,6	0,2	0,2	0	-0
0	-0	-0	0,6	0,1	0,4	0,4	1	0,5	-0	0,3	-0	-0	0,2	0,1	-0	0,3	-0	0,1
0,4	-0	0,2	0,8	0,7	0,4	0,7	0,5	1	-0	0,6	-0	0,1	0,1	0,4	0,1	0,3	-0	0,1
-0	0,1	-0	-0	-0	-0	-0	-0	-0	1	-0	0,2	0,2	-0	0	0	0,1	0,2	-0
0,3	-0	-0	0,3	0,4	0,2	0,2	0,3	0,6	-0	1	-0	0,2	0	0	-0	0,4	-0	0

-0	0,1	-0	-0	-0	-0	-0	-0	-0	0,2	-0	1	-0	-0	-0	0,1	0	0	-0
0,1	-0	-0	-0	0,1	-0	-0	-0	0,1	0,2	0,2	-0	1	-0	-0	-0	0,1	-0	0
-0	-0	0	0,2	0	0,2	0,1	0,2	0,1	-0	0	-0	-0	1	0,1	-0	-0	0	0
-0	-0	0,5	0,4	0,3	0	0,6	0,1	0,4	0	0	-0	-0	0,1	1	0,1	0,6	0	-0
0	0,5	0,2	0,1	0,3	0,1	0,2	-0	0,1	0	-0	0,1	-0	-0	0,1	1	-0	0,1	-0
-0	-0	-0	0,2	0,2	0,1	0,2	0,3	0,3	0,1	0,4	0	0,1	-0	0,6	-0	1	-0	0,1
-0	0,2	0	-0	-0	0	0	-0	-0	0,2	-0	0	-0	-0	0	0,1	-0	1	-1
0,2	-0	-0	0	0	-0	-0	0,1	0,1	-0	0	-0	0	0	-0	-0	0,1	-1	1

Tabla 0-22 Matriz de covarianzas Problema de regresión 3

1	0	0,48	0,19	0,8	0	0	0,01	0,43	-0,3	0,29	-0,2	0,14	-0	-0	0,02	-0	-0	0,04
0	1	-0,1	-0,3	-0	0,01	-0,2	-0,3	-0,4	0,08	-0,3	0,12	-0,2	-0,1	-0,1	0,45	-0,2	0,43	-0,4
0,48	-0,1	1	0,22	0,56	-0,1	0,32	-0,3	0,19	-0,1	-0,2	-0	-0,1	0,02	0,5	0,2	-0,1	-0	0,04
0,19	-0,3	0,22	1	0,51	0,44	0,83	0,57	0,81	-0,3	0,34	-0,1	-0,1	0,19	0,44	0,15	0,23	-0,2	0,23
0,8	-0	0,56	0,51	1	0,18	0,46	0,11	0,7	-0,2	0,4	-0,1	0,07	0,03	0,32	0,3	0,17	-0,1	0,08
0	0,01	-0,1	0,44	0,18	1	0,33	0,4	0,36	-0,2	0,21	-0,1	-0,1	0,23	0,05	0,11	0,06	0,06	-0,1
0	-0,2	0,32	0,83	0,46	0,33	1	0,43	0,71	-0,1	0,17	-0,2	-0,1	0,15	0,58	0,24	0,22	-0,2	0,2
0,01	-0,3	-0,3	0,57	0,11	0,4	0,43	1	0,52	-0,2	0,31	-0,1	-0	0,22	0,05	-0,2	0,32	-0,1	0,12
0,43	-0,4	0,19	0,81	0,7	0,36	0,71	0,52	1	-0,2	0,6	-0,1	0,1	0,13	0,37	0,11	0,34	-0,3	0,25
-0,3	0,08	-0,1	-0,3	-0,2	-0,2	-0,1	-0,2	-0,2	1	-0,1	0,22	0,22	-0,1	0,01	0,02	0,07	0,06	-0,1
0,29	-0,3	-0,2	0,34	0,4	0,21	0,17	0,31	0,6	-0,1	1	-0	0,2	0,02	0,01	-0	0,45	-0,2	0,2
-0,2	0,12	-0	-0,1	-0,1	-0,1	-0,2	-0,1	-0,1	0,22	-0	1	-0	-0,1	-0	0,12	0,01	0,04	-0
0,14	-0,2	-0,1	-0,1	0,07	-0,1	-0,1	-0	0,1	0,22	0,2	-0	1	-0,1	-0,2	-0,1	0,11	-0	0,01
-0	-0,1	0,02	0,19	0,03	0,23	0,15	0,22	0,13	-0,1	0,02	-0,1	-0,1	1	0,07	-0,2	-0	####	####
-0	-0,1	0,5	0,44	0,32	0,05	0,58	0,05	0,37	0,01	0,01	-0	-0,2	0,07	1	0,07	0,55	-0,2	0,17
0,02	0,45	0,2	0,15	0,3	0,11	0,24	-0,2	0,11	0,02	-0	0,12	-0,1	-0,2	0,07	1	-0,1	####	-0,1
-0	-0,2	-0,1	0,23	0,17	0,06	0,22	0,32	0,34	0,07	0,45	0,01	0,11	-0	0,55	-0,1	1	####	####
-0	0,43	-0	-0,2	-0,1	0,06	-0,2	-0,1	-0,3	0,06	-0,2	0,04	-0	####	-0,2	####	####	1	-1
0,04	-0,4	0,04	0,23	0,08	-0,1	0,2	0,12	0,25	-0,1	0,2	-0	0,01	####	0,17	-0,1	####	-1	1

Ahora adjuntamos tablas que muestran máximos, mínimos, medias y varianzas por provincias y por procesos electorales para el *Problema de regresión 1*.

Las siguientes tablas muestran en qué proceso electoral se alcanzaron los máximos y mínimos en cada provincia para cada clase.

Tabla 0-23 Procesos electorales máximos por provincia

PROVINCIA	MAX_REG_IZQ	MAX_REG_DER	MAX_CEN_IZQ	MAX_CEN_DER	MAX_IZQ	MAX_DER
Cuenca	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Ávila	2000,3	2019,11	2019,4	2019,4	2016,6	2000,3
Toledo	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Albacete	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11
León	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Guadalajara	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11

Zaragoza	2011,11	2008,3	2019,4	2019,4	2015,12	2000,3
Girona	2015,12	2011,11	2008,3	2015,12	2015,12	2011,11
Lleida	2019,4	2011,11	2008,3	2015,12	2015,12	2016,6
Cádiz	2000,3	2000,3	2004,3	2019,4	2015,12	2011,11
Alicante/Alacant	2019,4	2000,3	2019,4	2019,4	2015,12	2011,11
Madrid	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Asturias	2000,3	2011,11	2019,4	2019,4	2016,6	2000,3
Castellón/Castelló	2019,4	2000,3	2019,4	2019,4	2015,12	2011,11
Santa Cruz de Tenerife	2000,3	2000,3	2019,11	2019,4	2015,12	2011,11
Balears, Illes	2019,4	2000,3	2019,4	2019,4	2015,12	2011,11
Coruña, A	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Pontevedra	2000,3	2000,3	2019,4	2008,3	2015,12	2000,3
Palmas, Las	2019,11	2000,3	2019,4	2019,4	2015,12	2011,11
Navarra	2019,11	2000,3	2008,3	2004,3	2016,6	2000,3
Tarragona	2019,11	2011,11	2008,3	2015,12	2016,6	2011,11
Valencia/València	2019,4	2000,3	2019,4	2019,4	2015,12	2011,11
Gipuzkoa	2011,11	2004,3	2008,3	2015,12	2016,6	2000,3
Bizkaia	2011,11	2019,11	2008,3	2008,3	2016,6	2000,3
Araba/Álava	2011,11	2019,11	2008,3	2008,3	2016,6	2000,3
Barcelona	2019,4	2011,11	2008,3	2019,4	2015,12	2000,3
Cáceres	2000,3	2000,3	2019,4	2019,4	2011,11	2011,11
Ciudad Real	2000,3	2000,3	2019,4	2019,4	2011,11	2011,11
Badajoz	2000,3	2000,3	2019,4	2019,4	2011,11	2011,11
Melilla	2019,11	2000,3	2008,3	2008,3	2011,11	2011,11
Almería	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11
Murcia	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Zamora	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Palencia	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11
Salamanca	2000,3	2019,4	2019,4	2019,4	2015,12	2011,11
Segovia	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Ceuta	2000,3	2000,3	2019,4	2008,3	2015,12	2000,3
Huelva	2000,3	2000,3	2004,3	2019,4	2016,6	2011,11
Valladolid	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11
Jaén	2011,11	2000,3	2019,4	2019,4	2016,6	2011,11
Soria	2000,3	2019,4	2019,4	2019,4	2019,4	2000,3
Teruel	2008,3	2019,11	2019,4	2019,4	2015,12	2011,11
Granada	2000,3	2000,3	2019,4	2019,4	2015,12	2011,11
Rioja, La	2008,3	2000,3	2019,4	2019,4	2015,12	2011,11
Ourense	2008,3	2000,3	2019,4	2008,3	2015,12	2016,6
Málaga	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11
Burgos	2000,3	2000,3	2019,4	2019,4	2015,12	2000,3
Lugo	2000,3	2000,3	2019,4	2008,3	2015,12	2011,11

Cantabria	2019,11	2000,3	2019,4	2019,4	2015,12	2011,11
Huesca	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11
Sevilla	2019,11	2000,3	2019,4	2019,4	2016,6	2011,11
Córdoba	2000,3	2000,3	2019,4	2019,4	2016,6	2011,11

Tabla 0-24 Procesos electorales mínimos por provincia

PROVINCIA	MIN_REG_IZQ	MIN_REG_DER	MIN_CEN_IZQ	MIN_CEN_DER	MIN_IZQ	MIN_DER
Burgos	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Girona	2000,3	2000,3	2016,6	2000,3	2011,11	2019,11
Coruña, A	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Araba/Álava	2000,3	2016,6	2016,6	2000,3	2011,11	2019,4
Lleida	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Rioja, La	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Navarra	2000,3	2000,3	2016,6	2000,3	2011,11	2019,11
Asturias	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Gipuzkoa	2008,3	2011,11	2016,6	2000,3	2011,11	2019,11
Bizkaia	2000,3	2008,3	2016,6	2000,3	2011,11	2019,4
Ourense	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Teruel	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Segovia	2000,3	2000,3	2016,6	2000,3	2011,11	2019,4
Tarragona	2000,3	2000,3	2016,6	2000,3	2011,11	2019,11
Barcelona	2000,3	2015,12	2015,12	2000,3	2011,11	2019,4
León	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Cuenca	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Albacete	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Lugo	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Toledo	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Guadalajara	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Pontevedra	2011,11	2000,3	2000,3	2000,3	2011,11	2019,4
Zaragoza	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Soria	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Ceuta	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Badajoz	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Madrid	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Cáceres	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Huelva	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Almería	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Valladolid	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Jaén	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Zamora	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Castellón/Castelló	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4

Granada	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Ciudad Real	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Huesca	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Salamanca	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Alicante/Alacant	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Palencia	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Balears, Illes	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Murcia	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Santa Cruz de Tenerife	2000,3	2016,6	2011,11	2000,3	2011,11	2019,4
Cantabria	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Valencia/València	2000,3	2000,3	2011,11	2000,3	2011,11	2019,4
Ávila	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Palmas, Las	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Melilla	2000,3	2000,3	2000,3	2000,3	2011,11	2019,4
Málaga	2000,3	2000,3	2011,11	2000,3	2008,3	2019,4
Sevilla	2000,3	2000,3	2016,6	2000,3	2008,3	2019,4
Córdoba	2000,3	2000,3	2011,11	2000,3	2008,3	2019,4
Cádiz	2000,3	2000,3	2011,11	2000,3	2008,3	2019,4

Las siguientes tablas muestran en qué provincias se alcanzaron los máximos y mínimos en cada proceso electoral para cada clase.

Tabla 0-25 Provincias máximas por proceso electoral

ANNO_ELEC CION	MES_ELEC CION	MAX_REG _IZQ	MAX_REG _DER	MAX_CEN _IZQ	MAX_CEN_D ER	MAX_IZ Q	MAX_ DER
2000	3	Gipuzkoa	Girona	Sevilla	León	Córdoba	Ávila
2008	3	Navarra	Lleida	Tarragona	Valencia/Val ència	Soria	Murcia
2004	3	Girona	Bizkaia	Huelva	Navarra	Cádiz	Ávila
2016	6	Girona	Bizkaia	Sevilla	Tarragona	Araba/Ál ava	Lugo
2011	11	Gipuzkoa	Lleida	Sevilla	Navarra	Córdoba	Murcia
2015	12	Lleida	Bizkaia	Huelva	Tarragona	Barcelon a	Ávila
2019	11	Lleida	Bizkaia	Jaén	Teruel	Araba/Ál ava	Ávila
2019	4	Lleida	Bizkaia	Jaén	Madrid	Araba/Ál ava	Ávila

Tabla 0-26 Provincias mínimas por proceso electoral

ANNO_ELEC CION	MES_ELECC ION	MIN_REG _IZQ	MIN_REG_ DER	MIN_CEN _IZQ	MIN_CEN_D ER	MIN_I ZQ	MIN_D ER
-------------------	------------------	-----------------	-----------------	-----------------	-----------------	-------------	-------------

2011	11	León	León	Murcia	Valencia/València	Navarra	Girona
2019	11	León	León	Melilla	Valencia/València	Navarra	Gipuzkoa
2004	3	León	León	Ávila	Valencia/València	Navarra	Girona
2019	4	León	León	Melilla	Valencia/València	Navarra	Gipuzkoa
2000	3	León	León	Melilla	Valencia/València	Navarra	Girona
2008	3	León	León	Murcia	Valencia/València	Navarra	Girona
2015	12	León	León	Bizkaia	Valencia/València	Navarra	Barcelona
2016	6	León	León	Bizkaia	Valencia/València	Navarra	Gipuzkoa

Por último adjuntamos las medias por provincia y por proceso electoral para cada clase.

Tabla 0-27 Medias por provincia Problema de regresión 1

PROVINCIA	R_IZQ	R_DER	CEN_IZQ	CEN_DER	IZQ	DER
Valencia/València	0,00109	0,00013	0,30426	0,01011	0,09240	0,59201
Cantabria	0,02895	0,00000	0,24708	0,00300	0,00920	0,71177
Burgos	0,00000	0,00000	0,21468	0,00846	0,01200	0,76486
Ceuta	0,00000	0,00000	0,24807	0,00000	0,00134	0,75059
Tarragona	0,25925	0,17292	0,39685	0,01570	0,09681	0,05846
Segovia	0,00000	0,00000	0,15032	0,00917	0,00197	0,83854
Santa Cruz de Tenerife	0,00000	0,14491	0,37670	0,00238	0,04970	0,42631
Soria	0,00000	0,00147	0,19491	0,00346	0,00546	0,79470
Palencia	0,00000	0,00000	0,23909	0,00066	0,00071	0,75953
Balears, Illes	0,00031	0,00034	0,31676	0,01135	0,06735	0,60389
Murcia	0,00000	0,00009	0,13589	0,01716	0,00046	0,84639
Zaragoza	0,00051	0,00248	0,45951	0,01928	0,03067	0,48755
Alicante/Alacant	0,00034	0,00000	0,29924	0,01421	0,03303	0,65318
Pontevedra	0,00147	0,00000	0,26101	0,00000	0,07148	0,66604
Salamanca	0,00000	0,00018	0,16235	0,01105	0,00094	0,82547
Teruel	0,00037	0,04529	0,31436	0,00868	0,00569	0,62561
Ciudad Real	0,00000	0,00000	0,44474	0,00177	0,00000	0,55348
Huesca	0,00092	0,00184	0,47145	0,00464	0,01578	0,50538
Granada	0,00033	0,00000	0,61567	0,00614	0,00728	0,37058
Ourense	0,01150	0,00000	0,18657	0,00000	0,00407	0,79785
Melilla	0,08962	0,02115	0,15607	0,00000	0,00000	0,73316
Castellón/Castelló	0,00017	0,00000	0,34160	0,00438	0,05187	0,60198
Bizkaia	0,02787	0,53204	0,25057	0,00000	0,13152	0,05800

Gipuzkoa	0,20548	0,35245	0,27378	0,00015	0,12430	0,04385
Asturias	0,00000	0,00108	0,48330	0,00423	0,05218	0,45921
Jaén	0,00013	0,00000	0,70172	0,00164	0,00397	0,29253
Zamora	0,00000	0,00000	0,20560	0,00065	0,00034	0,79340
Palmas, Las	0,00401	0,03623	0,35105	0,00464	0,06635	0,53771
Valladolid	0,00000	0,00000	0,30929	0,01725	0,00370	0,66975
Córdoba	0,00014	0,00000	0,62242	0,00438	0,02254	0,35053
Almería	0,00000	0,00000	0,38211	0,00629	0,00016	0,61143
Huelva	0,00000	0,00000	0,69876	0,00926	0,00295	0,28903
Navarra	0,11287	0,00056	0,21039	0,00026	0,10689	0,56902
Cáceres	0,00000	0,00000	0,57588	0,00365	0,00000	0,42047
Madrid	0,00000	0,00000	0,33097	0,02784	0,04243	0,59876
Barcelona	0,16591	0,15626	0,49130	0,00283	0,16586	0,01785
Badajoz	0,00000	0,00000	0,61591	0,00866	0,00000	0,37543
Rioja, La	0,00025	0,00000	0,26782	0,00430	0,00495	0,72269
Cádiz	0,00000	0,00000	0,56032	0,01840	0,04574	0,37554
Málaga	0,00000	0,00000	0,51650	0,02247	0,01112	0,44991
Guadalajara	0,00000	0,00000	0,32147	0,01152	0,01465	0,65235
Araba/Álava	0,03711	0,18983	0,31674	0,00000	0,18294	0,27338
Lleida	0,35886	0,37463	0,19994	0,00207	0,03345	0,03105
Coruña, A	0,00288	0,00000	0,26896	0,00024	0,05220	0,67572
Toledo	0,00000	0,00000	0,34044	0,00882	0,00439	0,64635
Lugo	0,00022	0,00000	0,20426	0,00000	0,00583	0,78968
Sevilla	0,00005	0,00000	0,73155	0,00568	0,01816	0,24456
Ávila	0,00000	0,00105	0,09168	0,00735	0,00284	0,89707
Girona	0,31401	0,38054	0,25894	0,00242	0,03383	0,01026
Cuenca	0,00000	0,00000	0,35249	0,00065	0,00097	0,64589
Albacete	0,00000	0,00000	0,42198	0,00845	0,00499	0,56458
León	0,00000	0,00000	0,39053	0,00373	0,00580	0,59994

Tabla 0-28 Medias por proceso electoral

ANNO	MES	R_IZQ	R_DER	CEN_IZQ	CEN_DER	IZQ	DER
2011	11	0.01928029458	0.06724255657	0.1533879503	0.00002731803211	0.0009443653567	0.7591175152
2019	11	0.06734023661	0.04935622432	0.5063158333	0.0003073522426	0.009043651953	0.3676367015
2004	3	0.009118153943	0.05571201267	0.4589947932	0.00003767366042	0.0006496358012	0.4754877307
2019	4	0.06778783773	0.03754427561	0.5885665523	0.03894886279	0.01274914952	0.254403322
2000	3	0.001840027216	0.07910413637	0.2473027553	0.00008012820513	0.001240380918	0.670432572
2008	3	0.003609540786	0.03675246195	0.488241243	0,00000	0.0003034488325	0.4710933054
2015	12	0.04862862423	0.01716495799	0.2066174679	0.01116226315	0.1270283881	0.5893982986
2016	6	0.03233287695	0.02873778572	0.1634773305	0.001661123858	0.1100866283	0.6637042547

H Gráficos de las regresiones lineales

Mostramos los gráficos obtenidos al realizar las regresiones lineales tanto mediante línea como mediante parábola. Dividimos estos gráficos en secciones que hacen referencia a cada problema de regresión considerado.

Problema de regresión 1

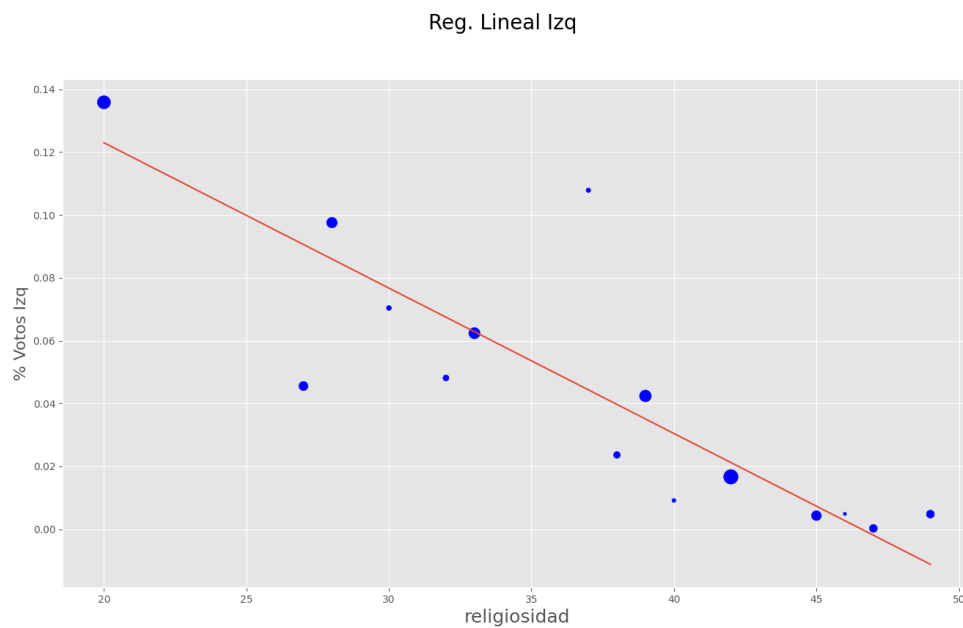


Figura 0-1 Regresión lineal mediante línea Problema de regresión 1, izquierda

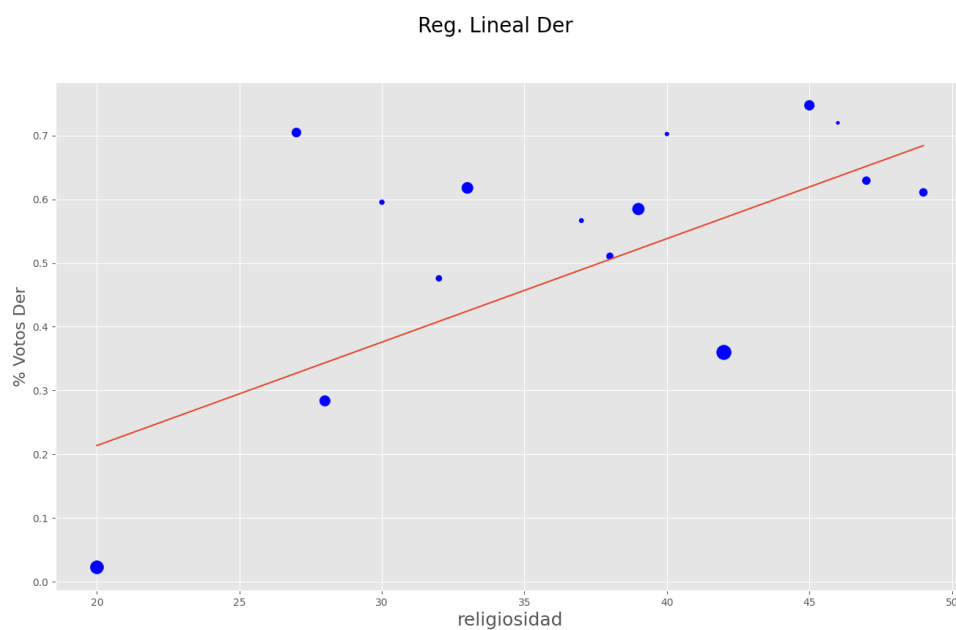


Figura 0-2 Regresión lineal mediante línea Problema de regresión 1, derecha

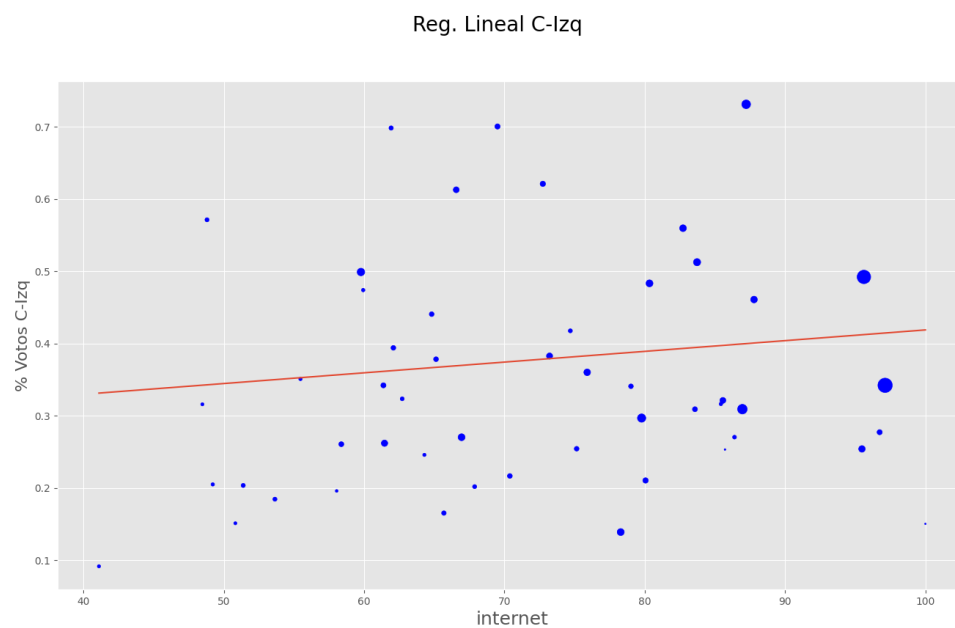


Figura 0-3 Regresión lineal mediante línea Problema de regresión 1, Centro-izquierda

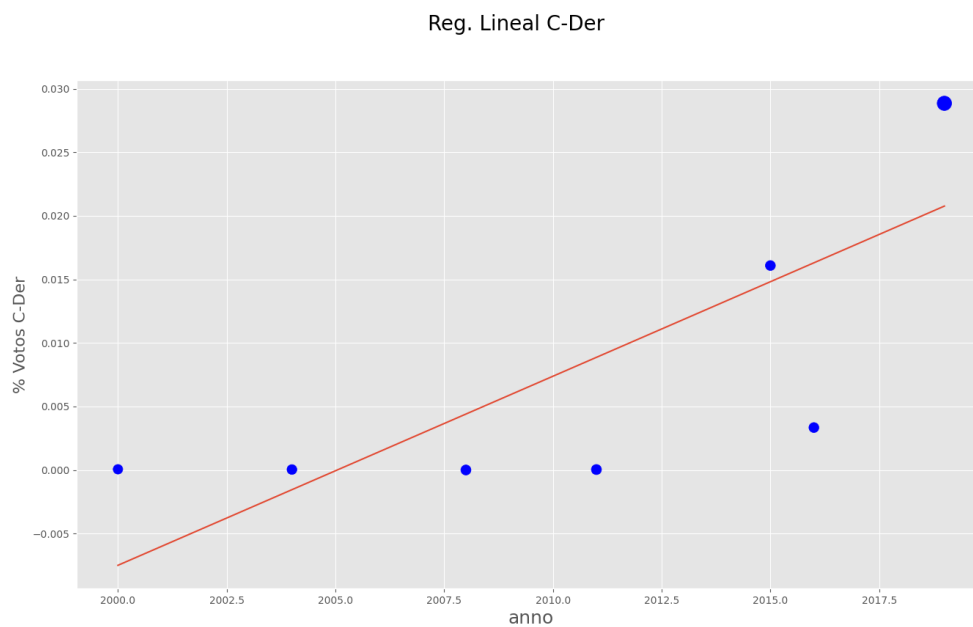


Figura 0-4 Regresión lineal mediante línea Problema de regresión 1, Centro-derecha

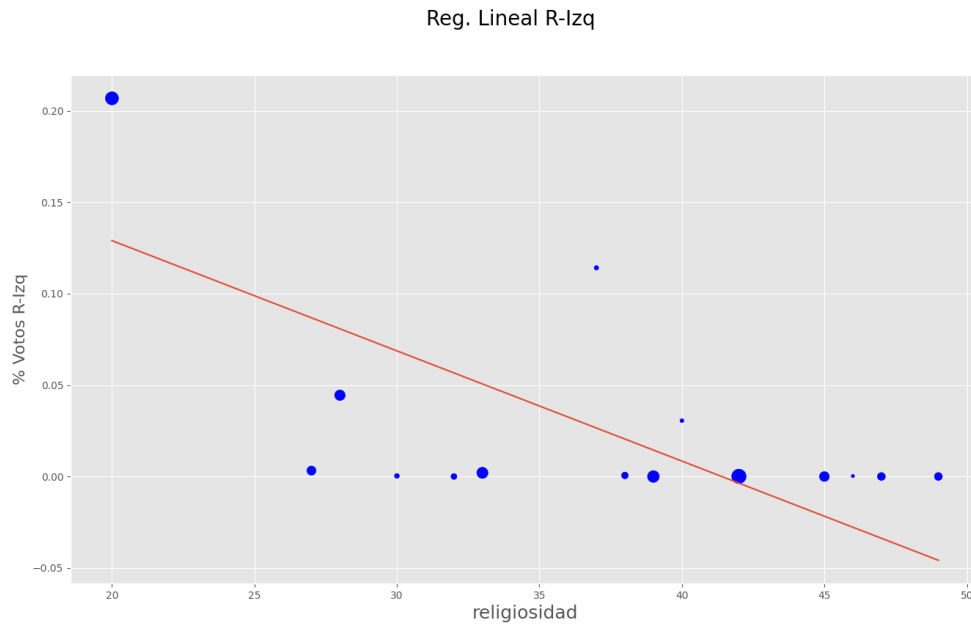


Figura 0-5 Regresión lineal mediante línea Problema de regresión 1, Regionalista-izquierda

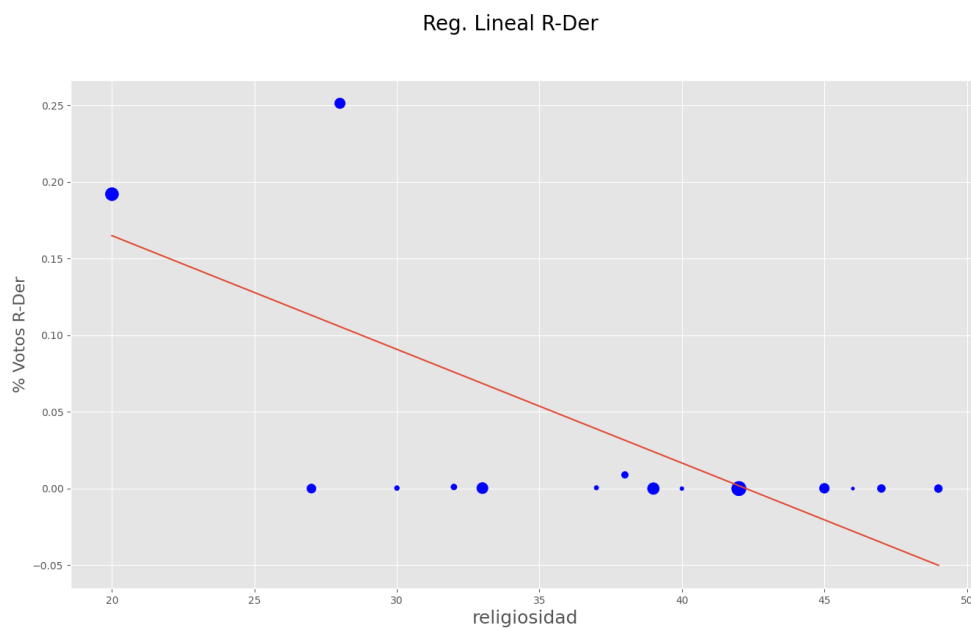


Figura 0-6 Regresión lineal mediante línea Problema de regresión 1, Regionalista-derecha

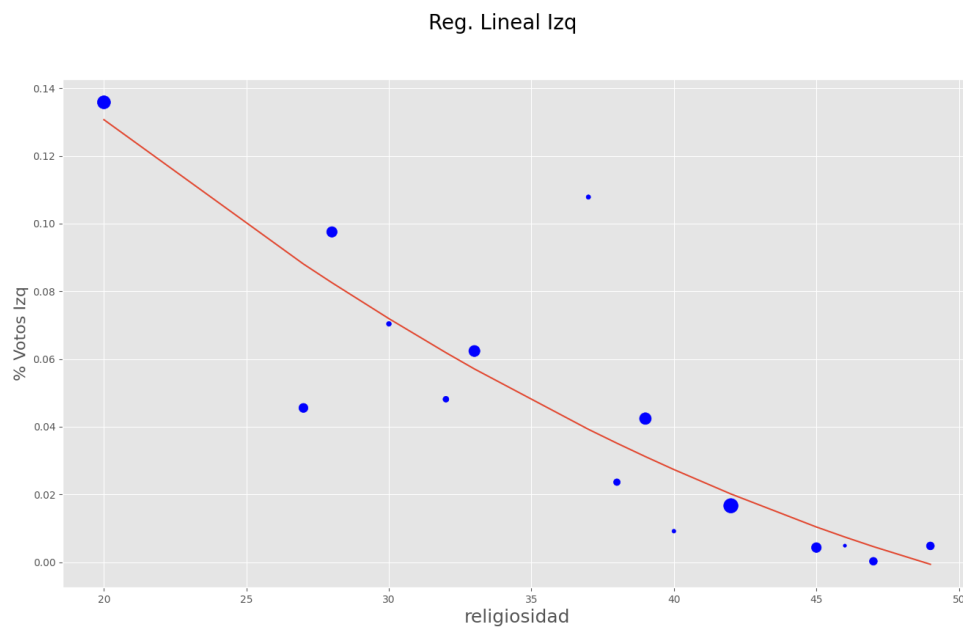


Figura 0-7 Regresión lineal mediante parábola Problema de regresión 1, Izquierda

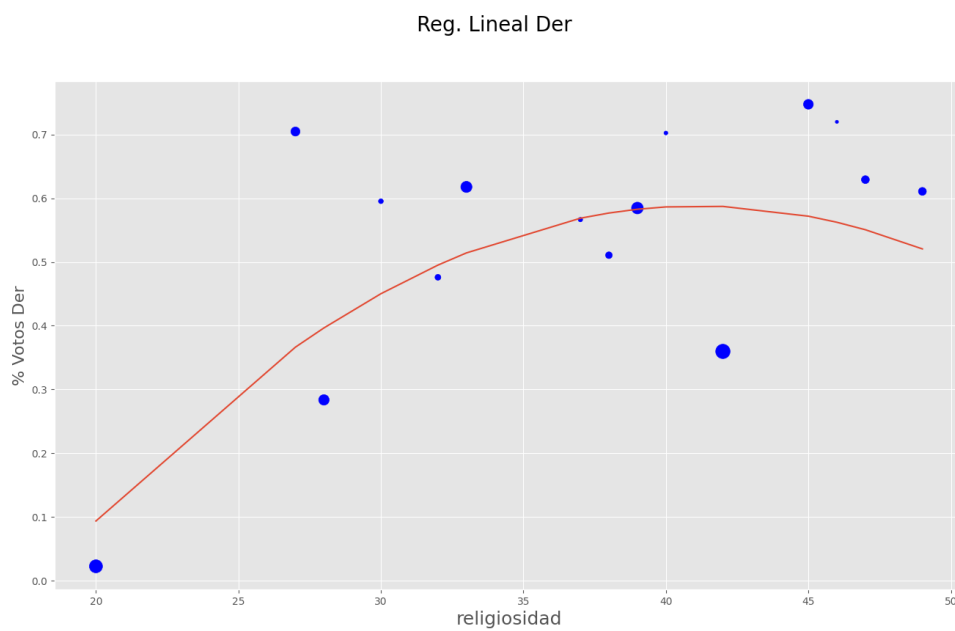


Figura 0-8 Regresión lineal mediante parábola Problema de regresión 1, Derecha

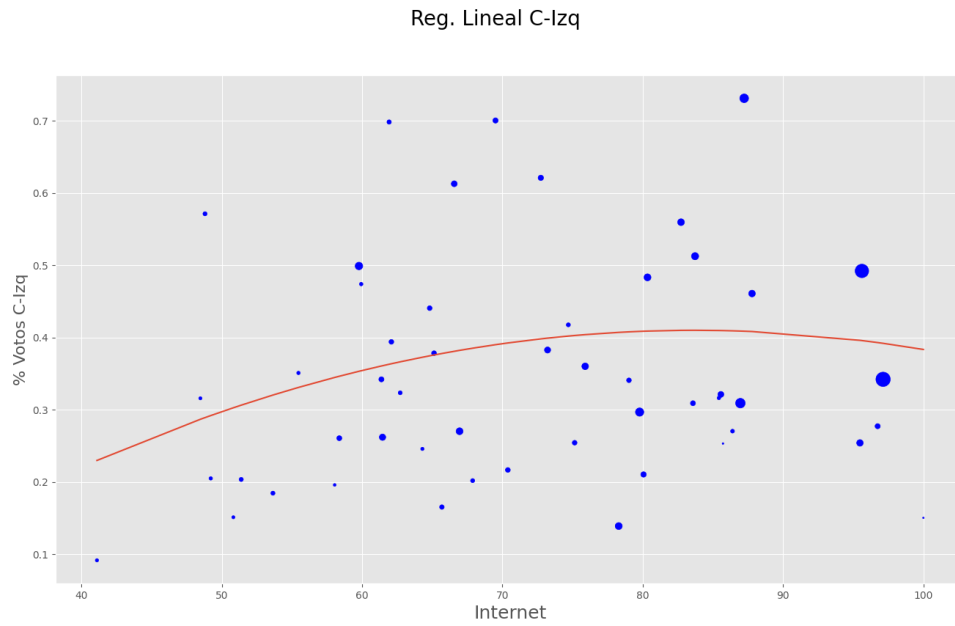


Figura 0-9 Regresión lineal mediante parábola Problema de regresión 1, Centro-izquierda

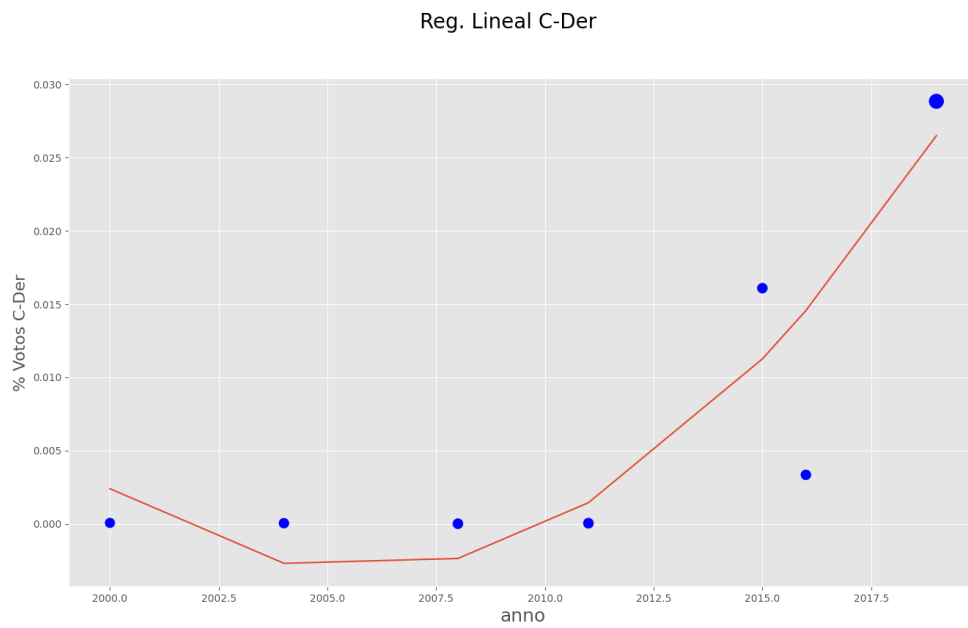


Figura 0-10 Regresión lineal mediante parábola Problema de regresión 1, Centro-derecha

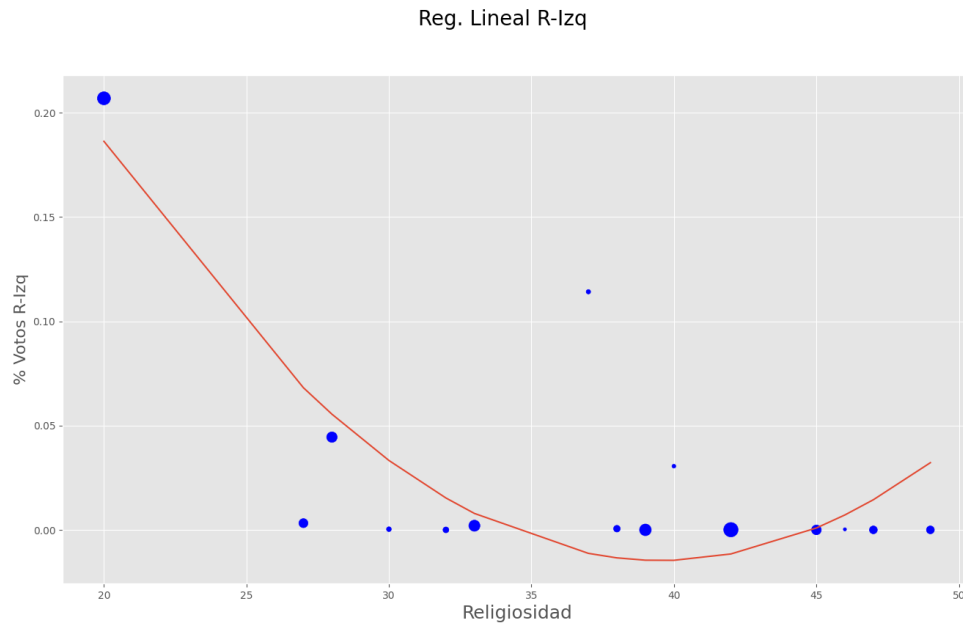


Figura 0-11 Regresión lineal mediante parábola Problema de regresión 1, Regionalista-izquierda

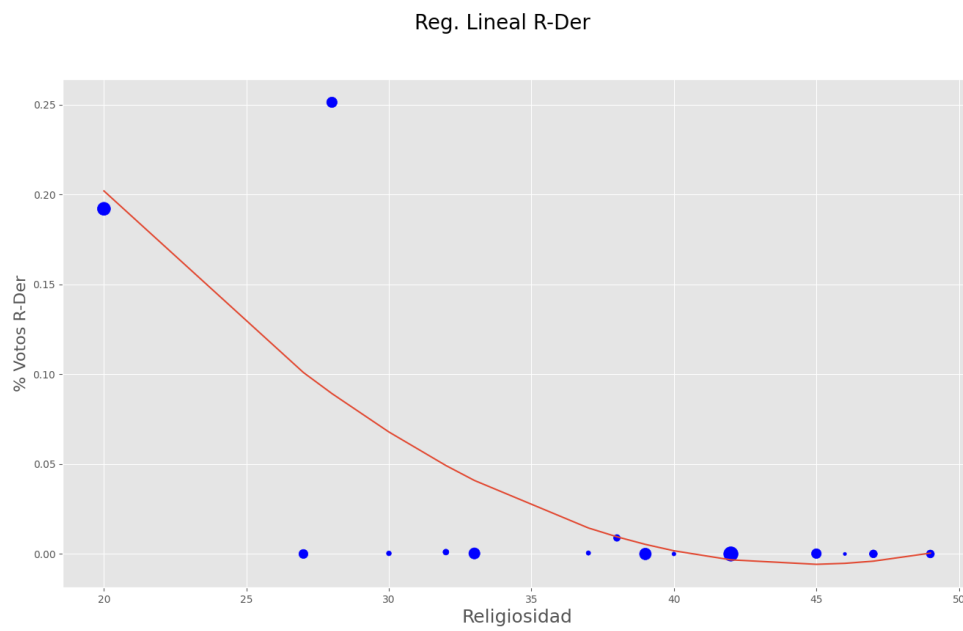


Figura 0-12 Regresión lineal mediante parábola Problema de regresión 1, Regionalista-derecha

Problema de regresión 2

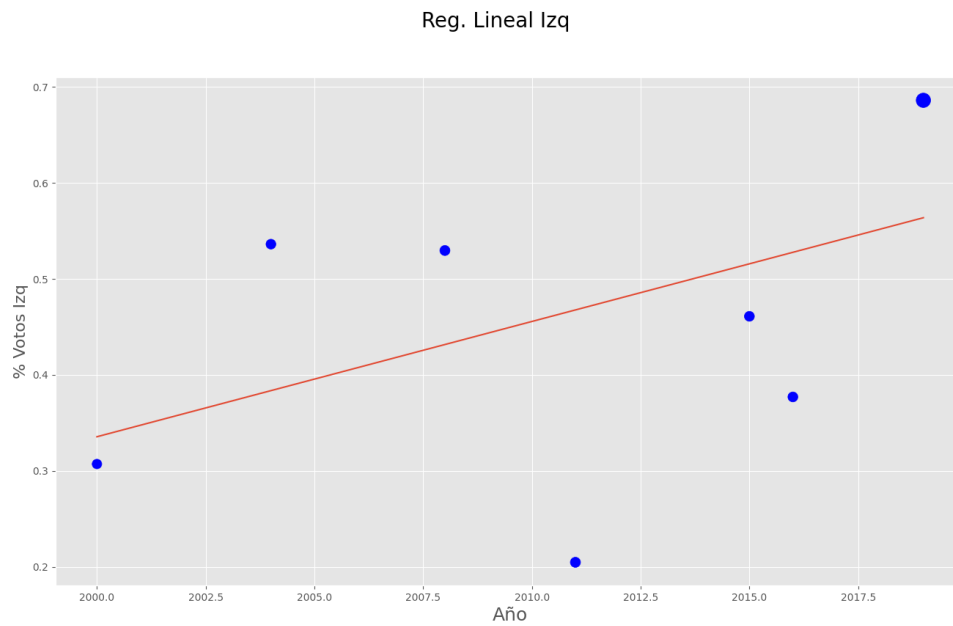


Figura 0-13 Regresión lineal mediante línea Problema de regresión 2, Izquierda

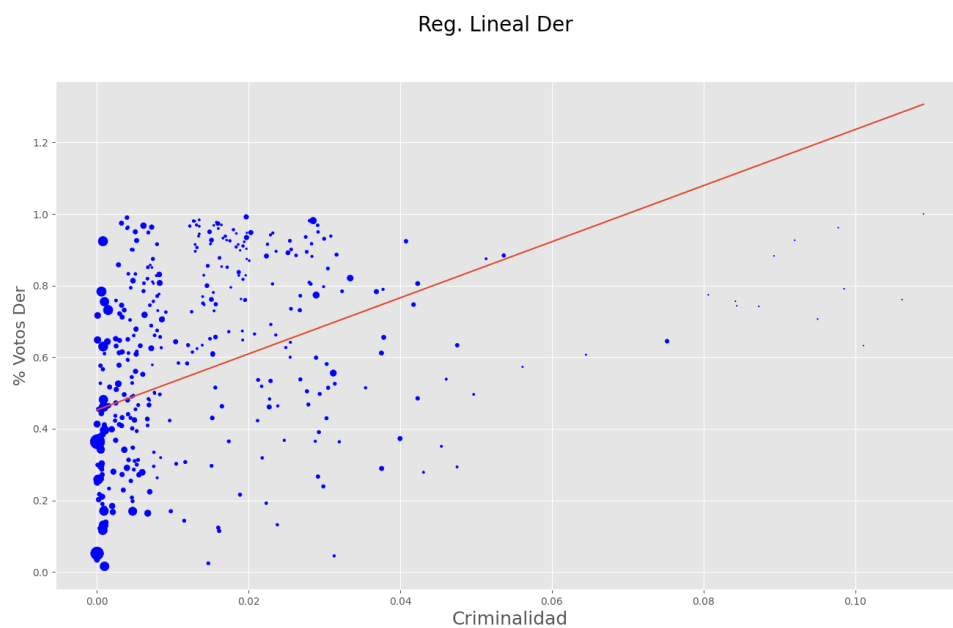


Figura 0-14 Regresión lineal mediante línea Problema de regresión 2, Derecha

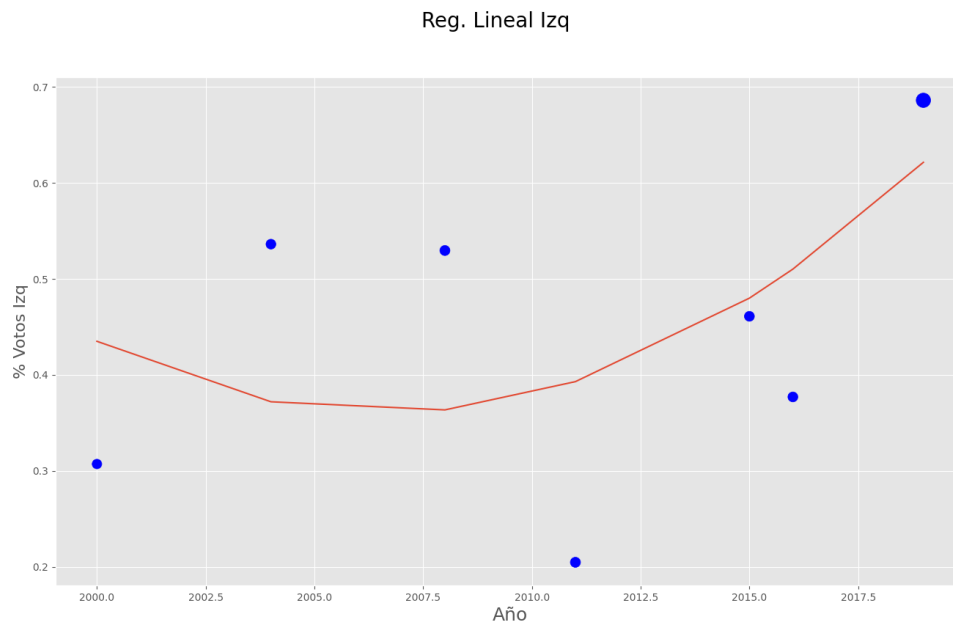


Figura 0-15 Regresión lineal mediante parábola Problema de regresión 2, Izquierda

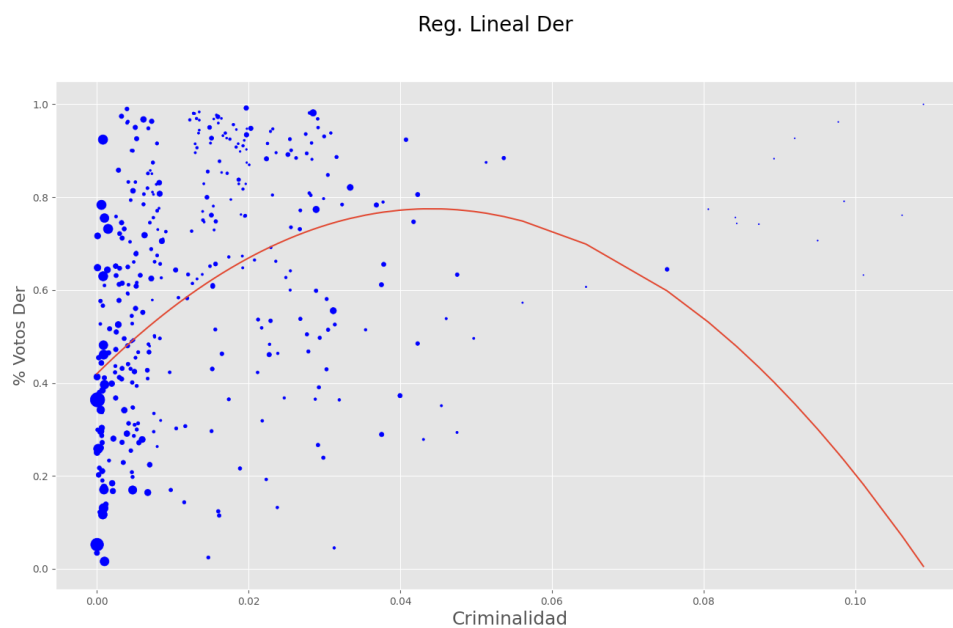


Figura 0-16 Regresión lineal mediante parábola Problema de regresión 2, Derecha

Problema de regresión 3

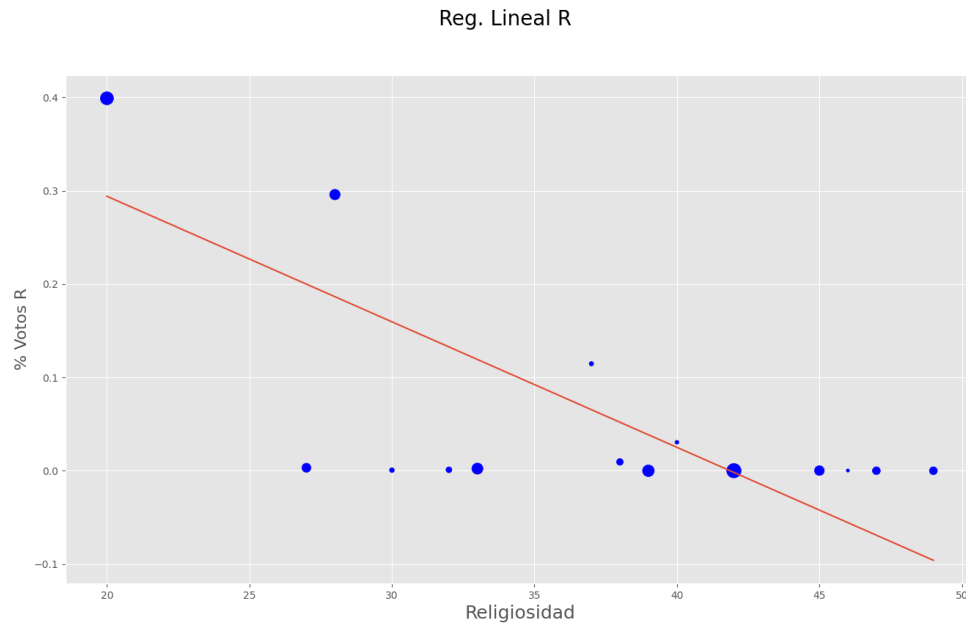


Figura 0-17 Regresión lineal mediante línea Problema de regresión 3, Regionalista

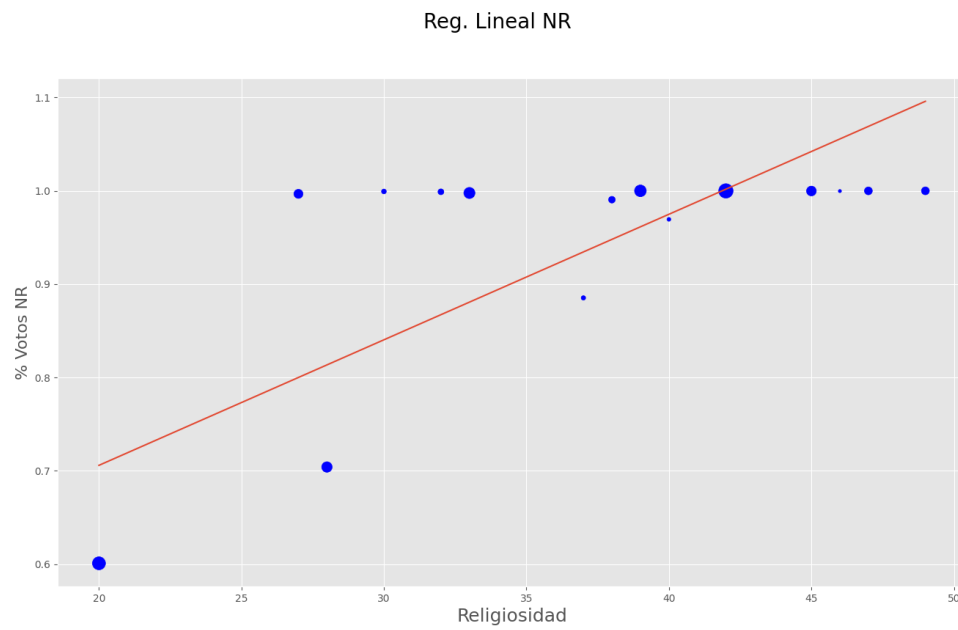


Figura 0-18 Regresión lineal mediante línea Problema de regresión 3, No regionalista

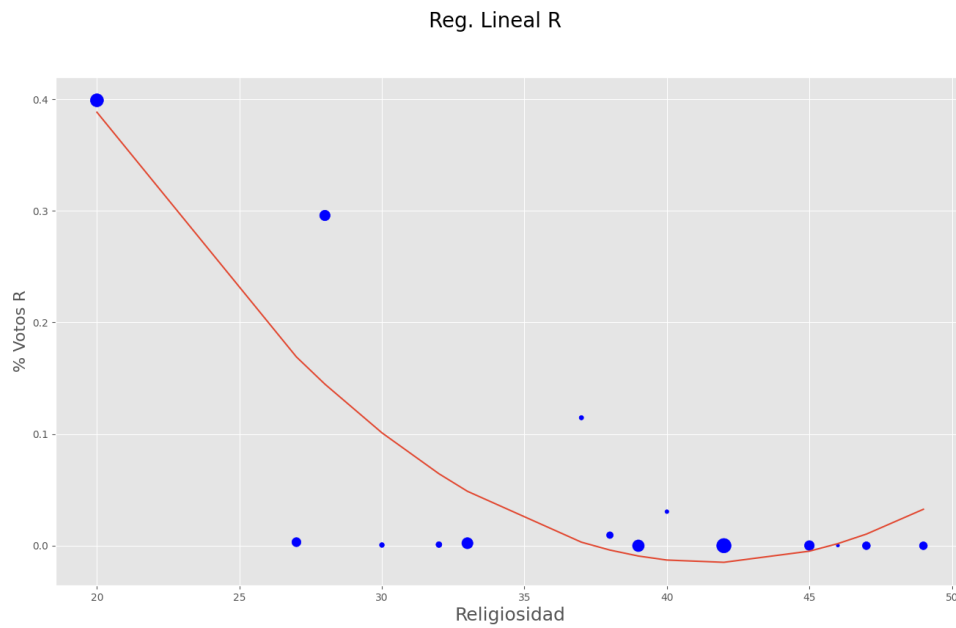


Figura 0-19 Regresión lineal mediante parábola Problema de regresión 3, Regionalista

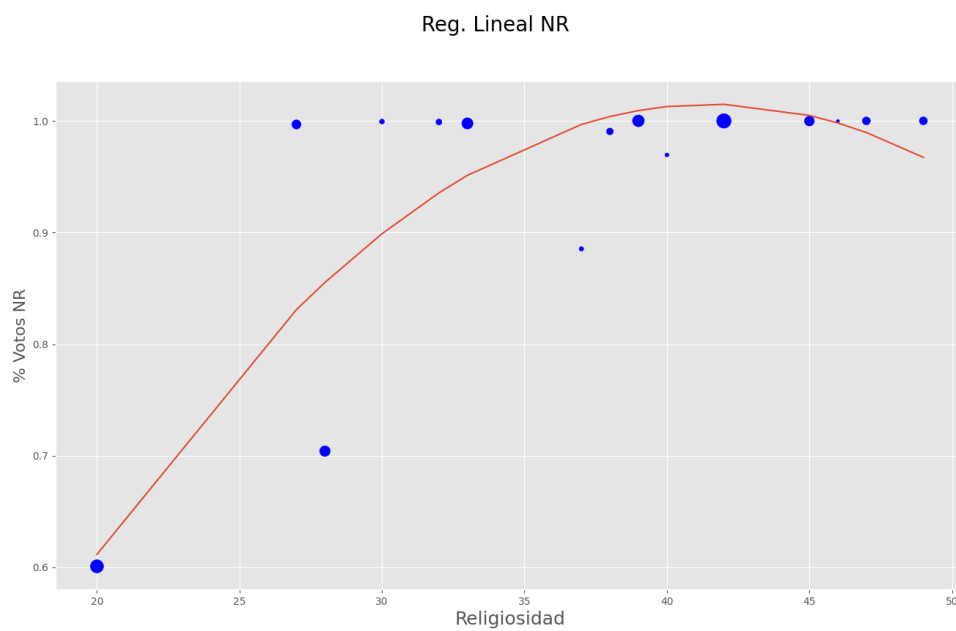


Figura 0-20 Regresión lineal mediante parábola Problema de regresión 3, No regionalista

Problema de regresión 4

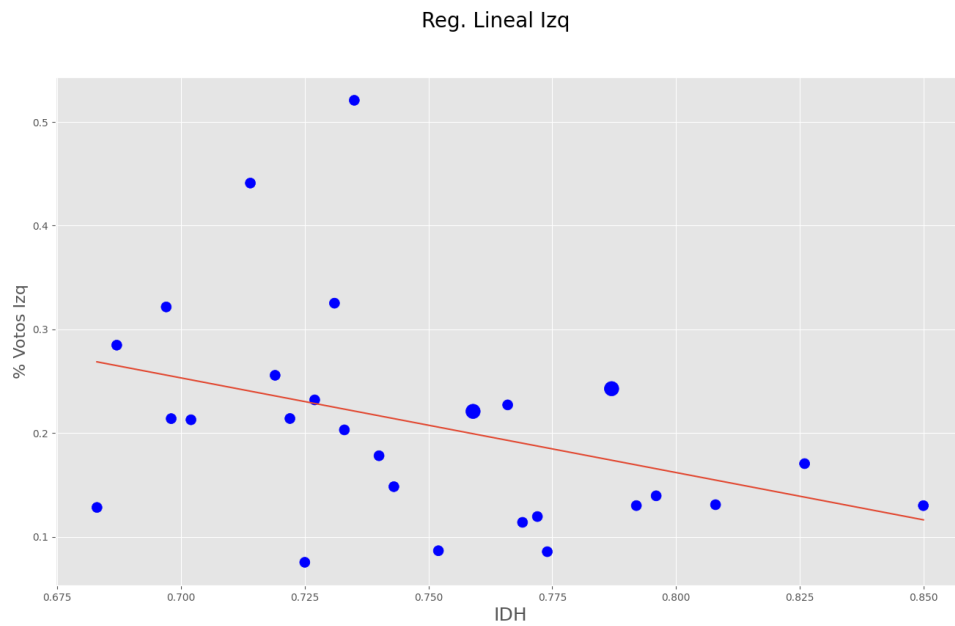


Figura 0-21 Regresión lineal mediante línea Problema de regresión 4, Brasil, Izquierda

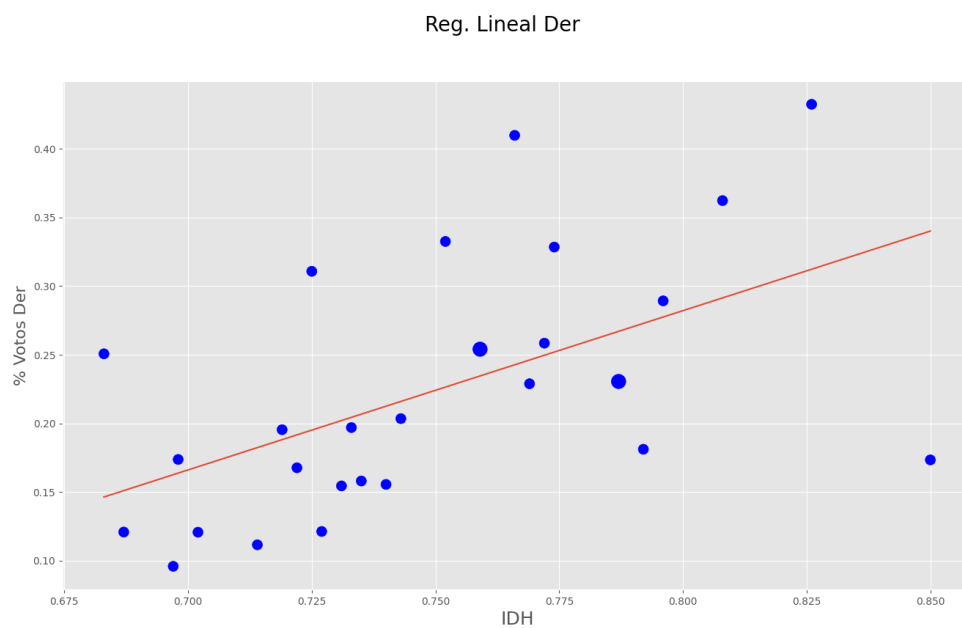


Figura 0-22 Regresión lineal mediante línea Problema de regresión 4, Brasil, Derecha

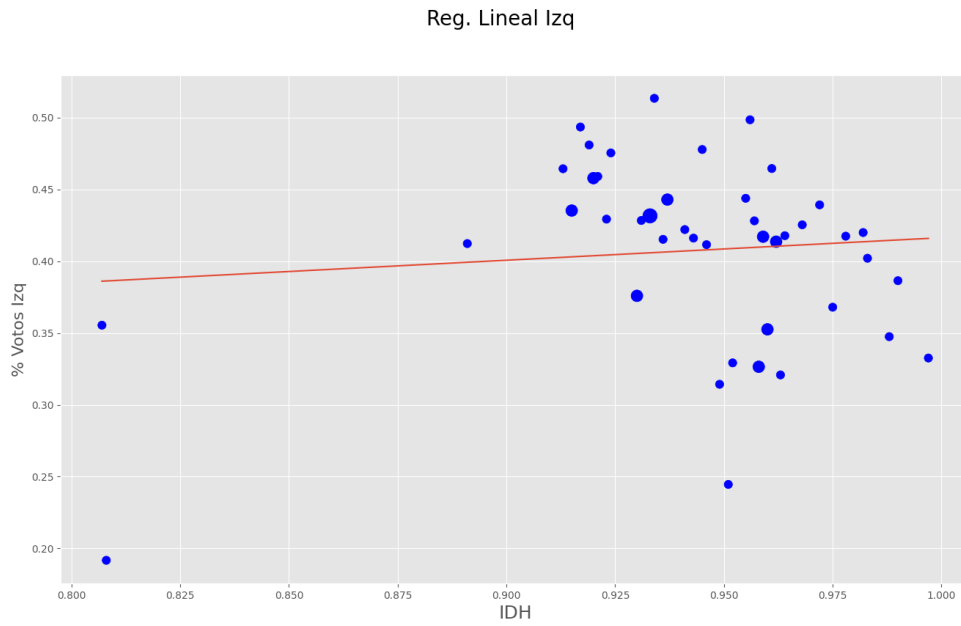


Figura 0-23 Regresión lineal mediante línea Problema de regresión 4, España, Izquierda

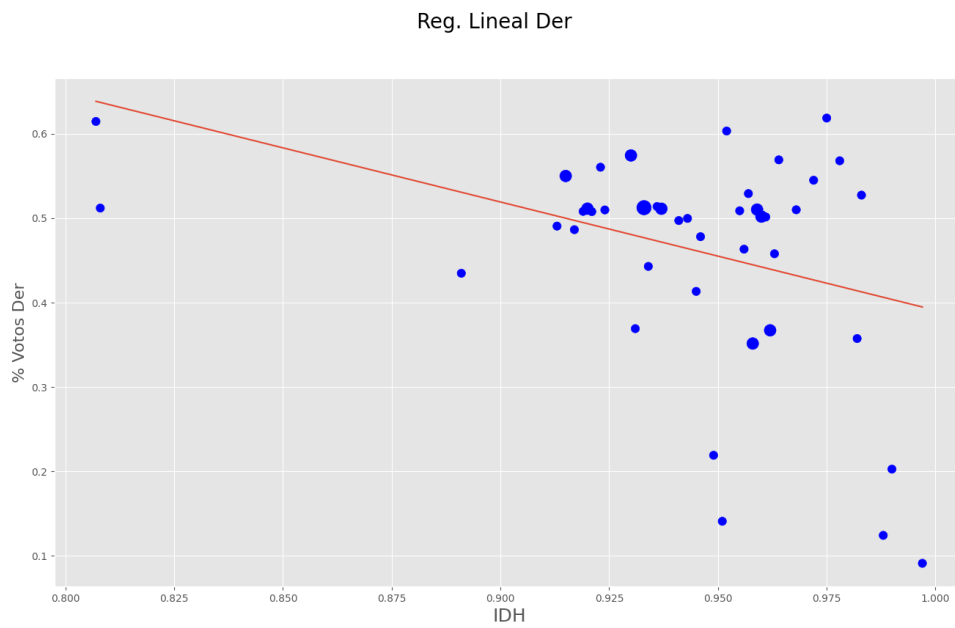


Figura 0-24 Regresión lineal mediante línea Problema de regresión 4, España, Derecha

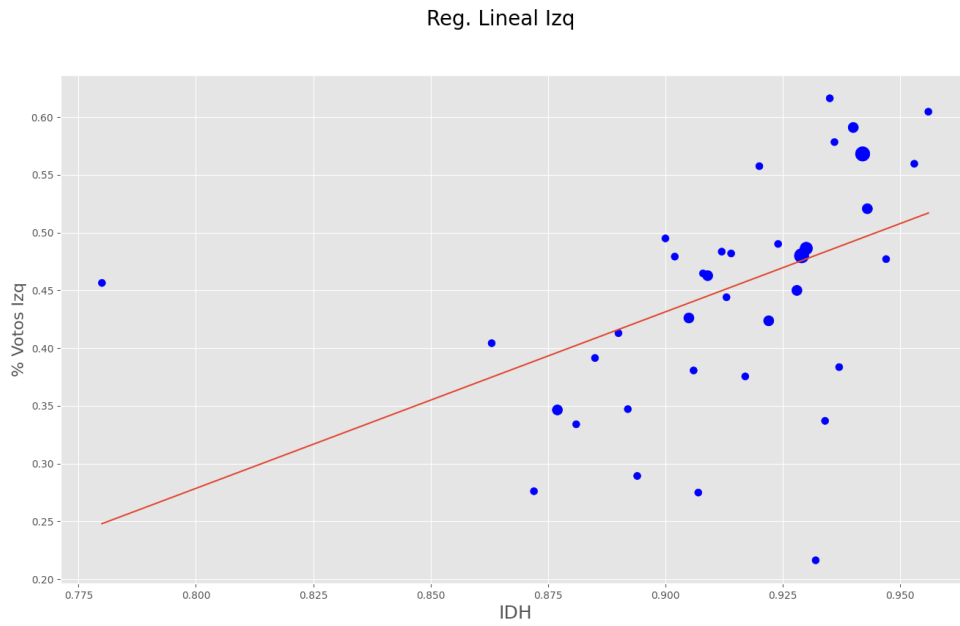


Figura 0-25 Regresión lineal mediante línea Problema de regresión 4, Estados Unidos, Izquierda

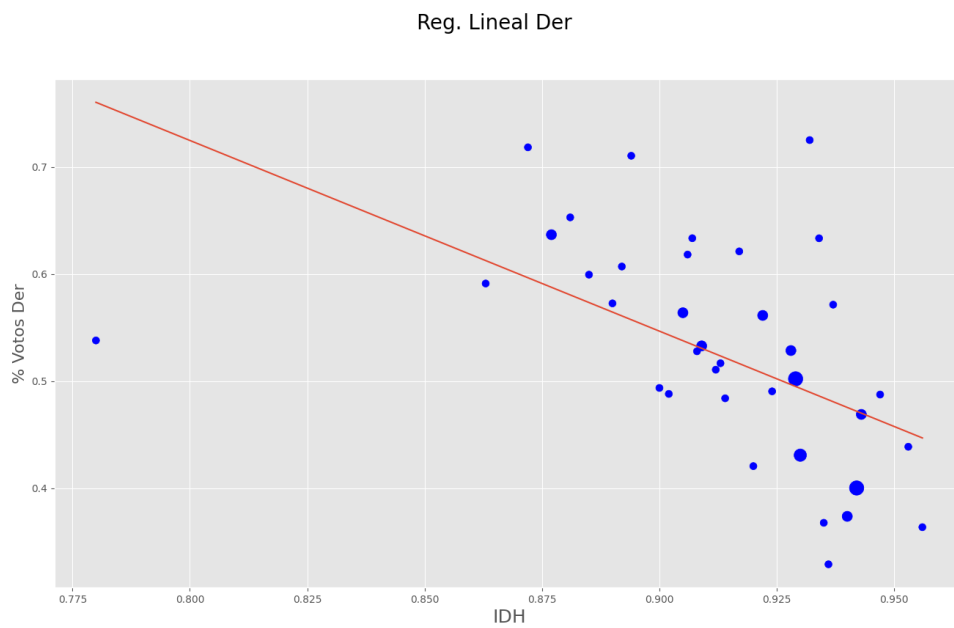


Figura 0-26 Regresión lineal mediante línea Problema de regresión 4, Estados Unidos, Derecha

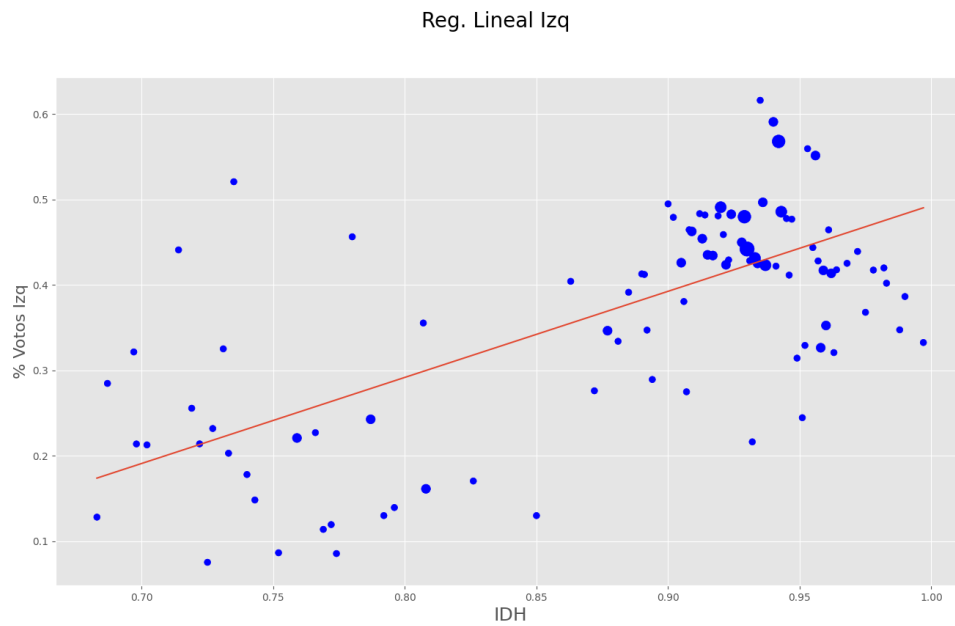


Figura 0-27 Regresión lineal mediante línea Problema de regresión 4, Brasil, España y Estados Unidos, Izquierda

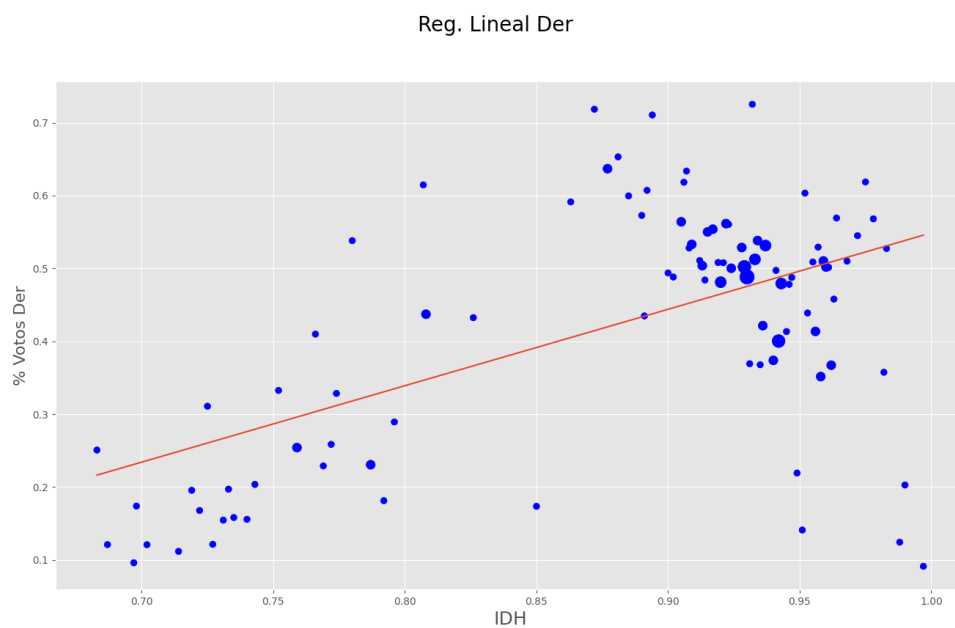


Figura 0-28 Regresión lineal mediante línea Problema de regresión 4, Brasil, España, Estados Unidos, Derecha

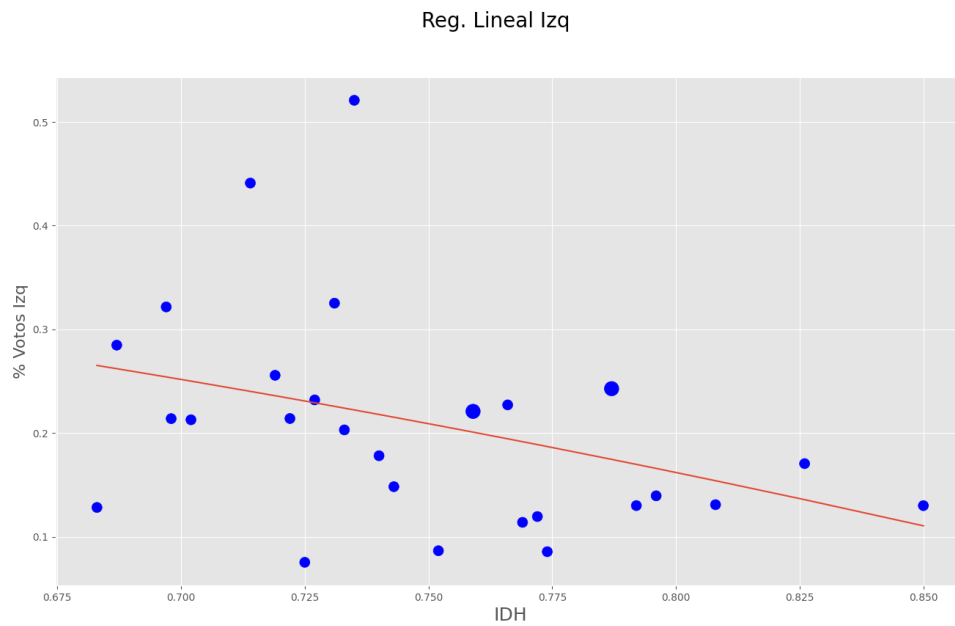


Figura 0-29 Regresión lineal mediante parábola Problema de regresión 4, Brasil, Izquierda

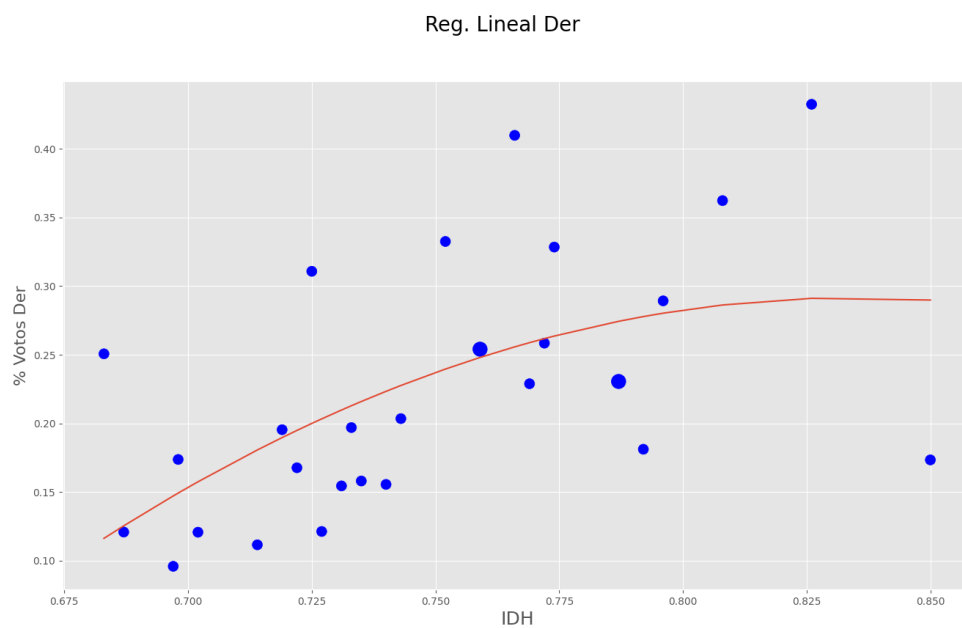


Figura 0-30 Regresión lineal mediante parábola Problema de regresión 4, Brasil, Derecha

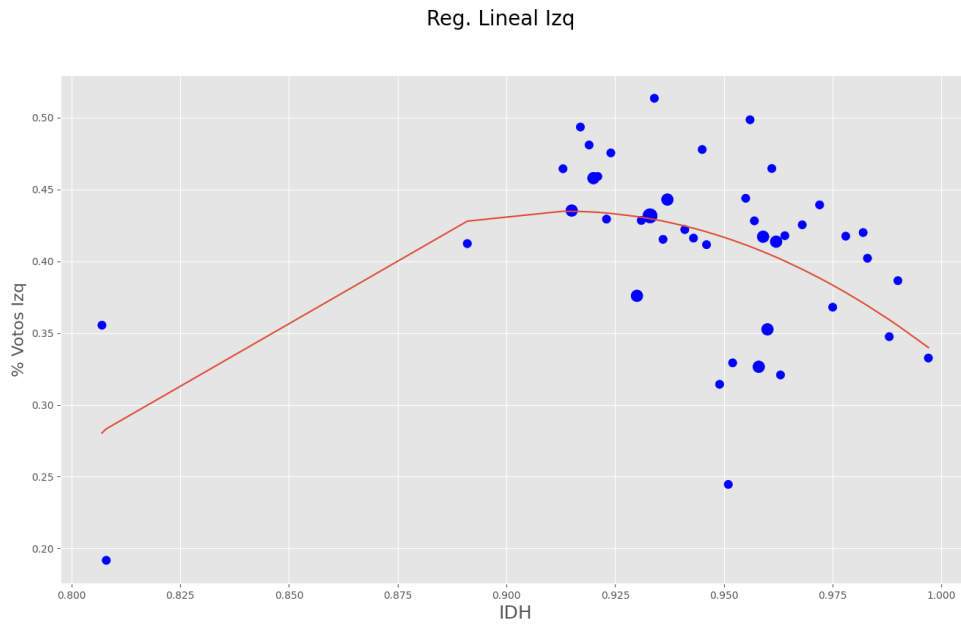


Figura 0-31 Regresión lineal mediante parábola Problema de regresión 4, España, Izquierda

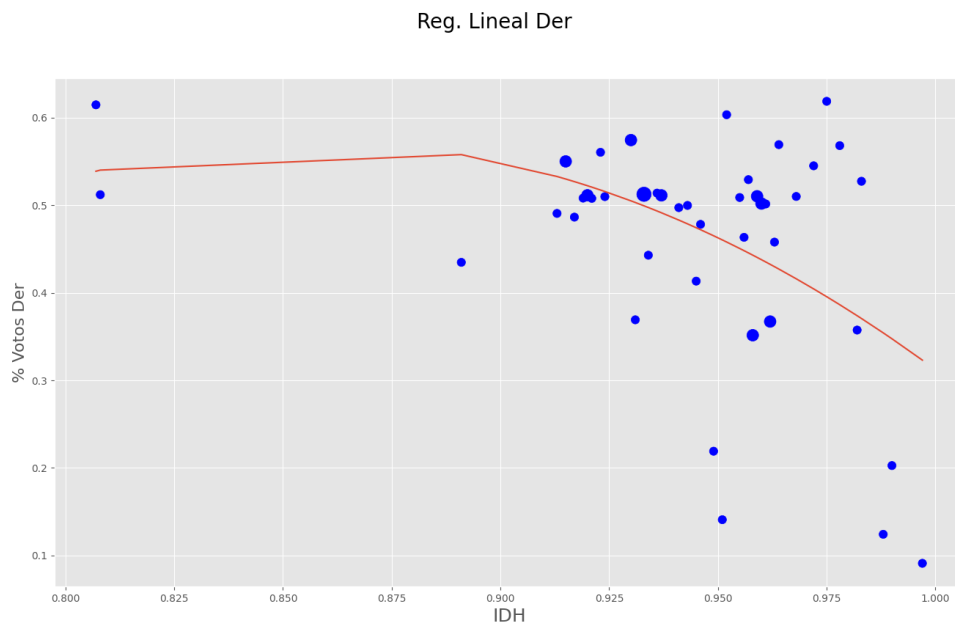


Figura 0-32 Regresión lineal mediante parábola Problema de regresión 4, España, Derecha

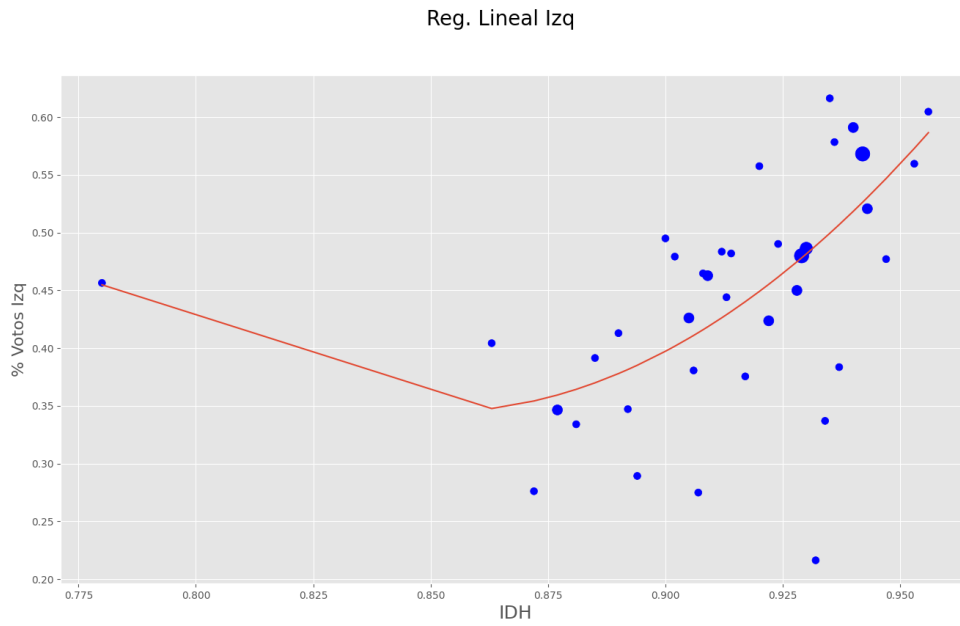


Figura 0-33 Regresión lineal mediante parábola Problema de regresión 4, Estados Unidos, Izquierda

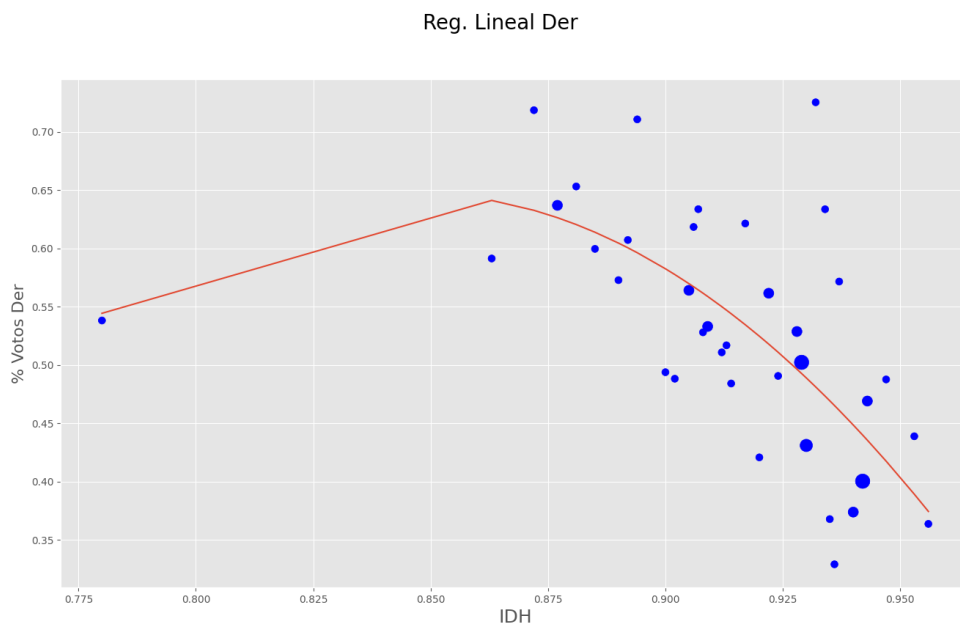


Figura 0-34 Regresión lineal mediante parábola Problema de regresión 4, Estados Unidos, Derecha

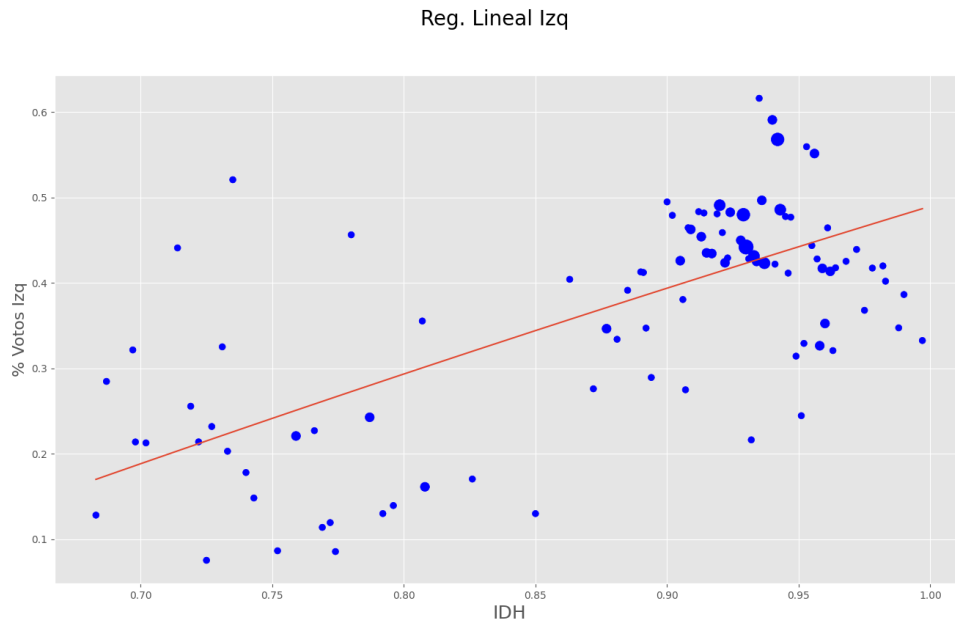


Figura 0-35 Regresión lineal mediante parábola Problema de regresión 4, Brasil, España y Estados Unidos, Izquierda

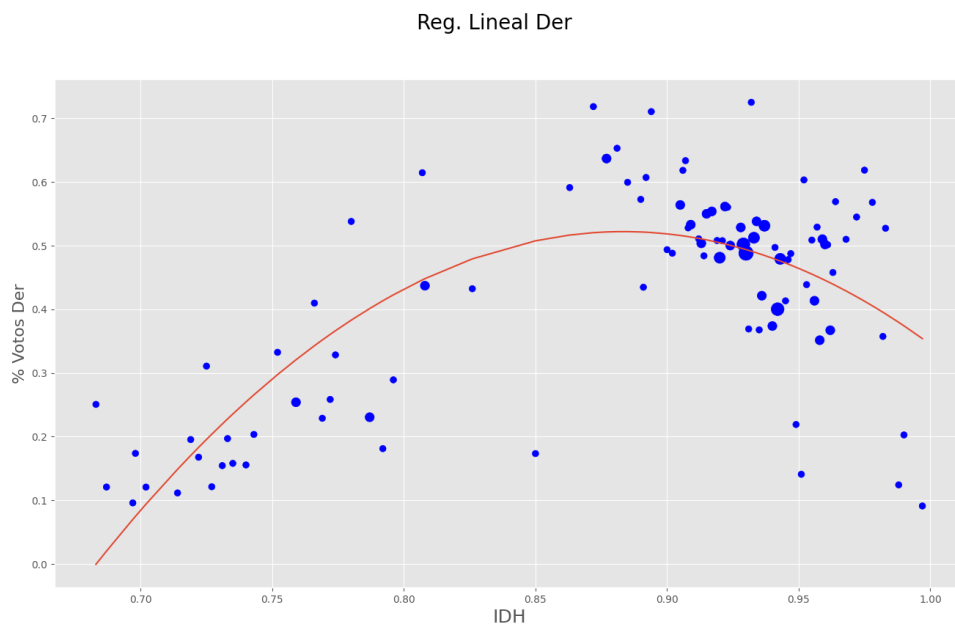


Figura 0-36 Regresión lineal mediante parábola Problema de regresión 4, Brasil, España y Estados Unidos, Derecha

I Tablas resultados Algoritmo KNN

Tabla 0-29 Resultados KNN Problema de clasificación 1

Distancia	Número de vecinos	Porcentaje de test	Accuracy
Manhattan	9	0,1	0,7417
Euclidea	9	0,1	0,7408
Chebyshev	9	0,1	0,7388
Manhattan	7	0,1	0,7375
Manhattan	9	0,15	0,7372
Euclidea	9	0,15	0,7369
Manhattan	9	0,25	0,7368
Chebyshev	9	0,15	0,7366
Euclidea	7	0,1	0,7365
Manhattan	9	0,2	0,7361
Euclidea	9	0,25	0,7359
Manhattan	7	0,15	0,7357
Chebyshev	9	0,2	0,7353
Chebyshev	7	0,1	0,7350
Euclidea	9	0,2	0,7350
Euclidea	7	0,15	0,7350
Chebyshev	9	0,25	0,7348
Chebyshev	7	0,15	0,7348
Manhattan	7	0,2	0,7343
Euclidea	7	0,2	0,7339
Manhattan	7	0,25	0,7339
Chebyshev	7	0,2	0,7335
Euclidea	7	0,25	0,7332
Chebyshev	7	0,25	0,7327
Manhattan	5	0,25	0,7309
Manhattan	5	0,1	0,7307
Manhattan	7	0,3	0,7303
Euclidea	5	0,25	0,7303
Chebyshev	5	0,25	0,7296
Euclidea	5	0,1	0,7294
Manhattan	9	0,3	0,7291
Chebyshev	7	0,3	0,7288
Euclidea	7	0,3	0,7288
Chebyshev	5	0,1	0,7285
Euclidea	9	0,3	0,7281
Chebyshev	9	0,3	0,7274
Manhattan	5	0,2	0,7272
Euclidea	5	0,2	0,7268
Chebyshev	5	0,2	0,7267
Manhattan	5	0,15	0,7224

Euclidean	5	0,15	0,7218
Chebyshev	5	0,15	0,7217
Manhattan	5	0,3	0,7216
Euclidean	5	0,3	0,7209
Chebyshev	5	0,3	0,7204
Manhattan	3	0,25	0,7170
Euclidean	3	0,25	0,7161
Chebyshev	3	0,25	0,7158
Manhattan	3	0,2	0,7100
Manhattan	3	0,1	0,7098
Euclidean	3	0,2	0,7098
Chebyshev	3	0,2	0,7094
Euclidean	3	0,1	0,7093
Chebyshev	3	0,1	0,7085
Manhattan	3	0,15	0,7058
Euclidean	3	0,15	0,7054
Chebyshev	3	0,15	0,7050
Manhattan	3	0,3	0,7046
Euclidean	3	0,3	0,7039
Chebyshev	3	0,3	0,7036

Tabla 0-30 Resultados KNN Problema de clasificación 2

Distancia	Número de vecinos	Porcentaje de test	Accuracy
Manhattan	9	0,15	0,7702
Euclidean	9	0,15	0,7701
Chebyshev	9	0,15	0,7691
Manhattan	7	0,15	0,7674
Manhattan	9	0,3	0,7673
Euclidean	7	0,15	0,7672
Euclidean	9	0,3	0,7666
Chebyshev	9	0,3	0,7664
Chebyshev	7	0,15	0,7664
Manhattan	9	0,25	0,7649
Euclidean	9	0,25	0,7644
Manhattan	9	0,2	0,7644
Chebyshev	9	0,25	0,7643
Euclidean	9	0,2	0,7637
Manhattan	5	0,15	0,7635
Chebyshev	9	0,2	0,7634
Euclidean	5	0,15	0,7628
Chebyshev	5	0,15	0,7626

Manhattan	9	0,1	0,7624
Manhattan	7	0,2	0,7624
Chebyshev	9	0,1	0,7620
Euclidean	9	0,1	0,7619
Manhattan	5	0,2	0,7619
Manhattan	7	0,25	0,7618
Euclidean	7	0,2	0,7615
Chebyshev	7	0,2	0,7615
Euclidean	5	0,2	0,7614
Chebyshev	5	0,2	0,7613
Manhattan	7	0,3	0,7612
Chebyshev	7	0,25	0,7611
Euclidean	7	0,25	0,7610
Chebyshev	7	0,3	0,7605
Euclidean	7	0,3	0,7604
Manhattan	7	0,1	0,7603
Euclidean	7	0,1	0,7600
Chebyshev	7	0,1	0,7589
Manhattan	5	0,3	0,7562
Chebyshev	5	0,3	0,7560
Euclidean	5	0,3	0,7558
Manhattan	5	0,1	0,7550
Euclidean	5	0,1	0,7545
Chebyshev	5	0,1	0,7542
Manhattan	3	0,15	0,7533
Euclidean	3	0,15	0,7528
Chebyshev	3	0,15	0,7527
Manhattan	3	0,3	0,7506
Euclidean	3	0,3	0,7504
Manhattan	3	0,2	0,7501
Chebyshev	3	0,3	0,7501
Euclidean	3	0,2	0,7498
Euclidean	5	0,25	0,7497
Chebyshev	3	0,2	0,7497
Manhattan	5	0,25	0,7494
Chebyshev	5	0,25	0,7492
Manhattan	3	0,1	0,7468
Euclidean	3	0,1	0,7465
Chebyshev	3	0,1	0,7462
Manhattan	3	0,25	0,7419
Euclidean	3	0,25	0,7414
Chebyshev	3	0,25	0,7410

Tabla 0-31 Resultados KNN Problema de clasificación 3

Distancia	Número de vecinos	Porcentaje de test	Precision
Manhattan	3	0,3	0,7210
Manhattan	5	0,25	0,7209
Euclidea	3	0,3	0,7201
Manhattan	3	0,25	0,7192
Euclidea	5	0,25	0,7188
Euclidea	3	0,25	0,7181
Chebyshev	5	0,25	0,7174
Chebyshev	3	0,3	0,7168
Chebyshev	3	0,25	0,7161
Manhattan	7	0,25	0,7156
Chebyshev	7	0,25	0,7120
Euclidea	7	0,25	0,7116
Manhattan	5	0,3	0,7114
Manhattan	7	0,3	0,7113
Euclidea	5	0,3	0,7076
Manhattan	9	0,2	0,7074
Manhattan	9	0,25	0,7059
Euclidea	7	0,3	0,7056
Chebyshev	5	0,3	0,7054
Chebyshev	7	0,3	0,7048
Manhattan	7	0,1	0,7040
Manhattan	5	0,15	0,7030
Euclidea	5	0,15	0,7014
Euclidea	9	0,25	0,7013
Chebyshev	9	0,25	0,7013
Euclidea	9	0,2	0,7010
Euclidea	3	0,1	0,6999
Manhattan	3	0,1	0,6999
Chebyshev	5	0,15	0,6999
Chebyshev	9	0,2	0,6999
Euclidea	7	0,1	0,6979
Chebyshev	3	0,1	0,6974
Manhattan	9	0,15	0,6974
Chebyshev	7	0,1	0,6964
Manhattan	5	0,1	0,6949
Euclidea	9	0,15	0,6934
Euclidea	5	0,1	0,6927
Manhattan	3	0,15	0,6926
Euclidea	3	0,15	0,6919
Chebyshev	9	0,15	0,6914

Manhattan	7	0,15	0,6908
Chebyshev	3	0,15	0,6899
Euclidea	7	0,15	0,6893
Chebyshev	7	0,15	0,6893
Chebyshev	5	0,1	0,6891
Manhattan	5	0,2	0,6667
Euclidea	5	0,2	0,6632
Manhattan	9	0,3	0,6616
Chebyshev	5	0,2	0,6611
Manhattan	3	0,2	0,6590
Manhattan	9	0,1	0,6585
Euclidea	3	0,2	0,6582
Euclidea	9	0,3	0,6570
Chebyshev	3	0,2	0,6564
Chebyshev	9	0,3	0,6538
Manhattan	7	0,2	0,6535
Euclidea	9	0,1	0,6522
Chebyshev	9	0,1	0,6522
Euclidea	7	0,2	0,6480
Chebyshev	7	0,2	0,6466

J Resultados Red neuronal Single-layer

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,3 y utilizando el método de optimización Adam.

Tabla 0-32 Resultados Red neuronal Single-layer en función del número de épocas para el Problema de clasificación 1

Nº de épocas	Accuracy
10	0,66777
50	0,67034
100	0,68138
500	0,65662

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, un porcentaje de test de 0,3 y utilizando el método de optimización Adam.

Tabla 0-33 Resultados Red neuronal Single-layer en función del tamaño de lote para el Problema de clasificación 1

Tamaño de lote	Accuracy
10	0,62439
32	0,68517

62	0,68001
132	0,68313

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, tamaño de lote de 32 y utilizando el método de optimización Adam.

Tabla 0-34 Resultados Red neuronal Single-layer en función del porcentaje de test para el Problema de clasificación 1

Porcentaje de test	Accuracy
0.3	0,67717
0.25	0,67477
0.2	0,65916
0.15	0,68797
0.1	0,67689

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, tamaño de lote de 32, porcentaje de test de 0,15 y utilizando el método de optimización RMSProp.

Tabla 0-35 Resultados Red neuronal Single-layer en función de la tasa de aprendizaje y momentum para el Problema de clasificación 1

Learning rate	Momentum	Accuracy
0.1	0	0,48469
0.1	0.5	0,46210
0.1	0.9	0,64611
0.1	0.99	0,56071
0.01	0	0,57310
0.01	0.5	0,64232
0.01	0.9	0,57020
0.01	0.99	0,58571
0.001	0	0,65415
0.001	0.5	0,67329
0.001	0.9	0,47119
0.001	0.99	0,62875
0,00001	0	0,67706
0,00001	0.5	0,67189
0,00001	0.9	0,62017
0,00001	0.99	0,56551

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,3 y utilizando el método de optimización Adam.

Tabla 0-36 Resultados Red neuronal Single-layer en función del número de épocas para el Problema de clasificación 2

Nº de épocas	Accuracy
10	0,68616
50	0,71148
100	0,71389
500	0,71435

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, un porcentaje de test de 0,3 y utilizando el método de optimización Adam.

Tabla 0-37 Resultados Red neuronal Single-layer en función del tamaño de lote para el Problema de clasificación 2

Tamaño de lote	Accuracy
10	0,70682
32	0,69873
62	0,71564
132	0,71682

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, tamaño de lote de 62 y utilizando el método de optimización Adam.

Tabla 0-38 Resultados Red neuronal Single-layer en función del porcentaje de test para el Problema de clasificación 2

Porcentaje de test	Accuracy
0.3	0,71791
0.25	0,71704
0.2	0,71933
0.15	0,70059
0.1	0,71669

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, tamaño de lote de 62, porcentaje de test de 0,2 y utilizando el método de optimización RMSProp.

Tabla 0-39 Resultados Red neuronal Single-layer en función de la tasa de aprendizaje y momentum para el Problema de clasificación 2

Learning rate	Momentum	Accuracy
0.1	0	0,71662
0.1	0.5	0,67397
0.1	0.9	0,64861

0.1	0.99	0,47903
0.01	0	0,69631
0.01	0.5	0,71309
0.01	0.9	0,71514
0.01	0.99	0,58623
0.001	0	0,71593
0.001	0.5	0,69282
0.001	0.9	0,62550
0.001	0.99	0,56906
0,00001	0	0,70902
0,00001	0.5	0,69457
0,00001	0.9	0,70980
0,00001	0.99	0,70587

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,3 y utilizando el método de optimización Adam.

Tabla 0-40 Resultados Red neuronal Single-layer en función del número de épocas para el Problema de clasificación 3

Nº de épocas	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
10	0,00000	0	0,92633
50	0,00000	0	0,93331
100	1,00000	0	0,93304
500	0,00000	0	0,93427

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, un porcentaje de test de 0,3 y utilizando el método de optimización Adam.

Tabla 0-41 Resultados Red neuronal Single-layer en función del número del tamaño de lote para el Problema de clasificación 3

Tamaño de lote	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
10	0,00000	0	0,93263
32	0,00000	0	0,93326
62	0,00000	0	0,93251
132	0,00000	0	0,93304

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, tamaño de lote de 32 y utilizando el método de optimización Adam.

Tabla 0-42 Resultados Red neuronal Single-layer en función del porcentaje de test para el Problema de clasificación 3

Porcentaje de test	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
0.3	0,00000	0	0,91857
0.25	0,00000	0	0,93337
0.2	0,00000	0	0,93242
0.15	0,00000	0	0,93099
0.1	1,00000	1	0,93383

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de épocas de 100, tamaño de lote de 32, porcentaje de test de 0,1 y utilizando el método de optimización *RMSPProp*.

Tabla 0-43 Resultados Red neuronal Single-layer en función de la tasa de aprendizaje y el momentum para el Problema de clasificación 3

Learning rate	Momentum	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
0.1	0	0,00000	0	0,93111
0.1	0.5	0,00000	0	0,86936
0.1	0.9	0,00000	0	0,91487
0.1	0.99	0,00000	0	0,91278
0.01	0	1,00000	1	0,87145
0.01	0.5	0,00000	0	0,91721
0.01	0.9	0,00000	0	0,84714
0.01	0.99	0,00000	0	0,93197
0.001	0	0,00000	0	0,92614
0.001	0.5	0,00000	0	0,86414
0.001	0.9	0,00000	0	0,89588
0.001	0.99	0,00000	0	0,92277
0,00001	0	0,00000	0	0,93499
0,00001	0.5	0,00000	0	0,91967
0,00001	0.9	0,00000	0	0,90319
0,00001	0.99	0,00000	0	0,93475

K Resultados Red neuronal Multi-layer

En este anexo se adjuntan las tablas que contienen los resultados obtenidos para las diferentes combinaciones de parámetros utilizadas en la ejecución de las Redes neuronales Multi-Layer.

En primer lugar mostramos los resultados obtenidos para el *Problema de clasificación 1*.

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam.

Tabla 0-44 Resultados Red neuronal Multi-layer en función del número de neuronas para el Problema de clasificación 1

Nº de neuronas	Accuracy
6	0,46388
11	0,63692
16	0,46388
21	0,46388
26	0,68077
31	0,62271
26	0,61798
41	0,67218
46	0,62273
51	0,66357
56	0,62269
71	0,68278
76	0,68451
81	0,68506
86	0,66643
91	0,68445
96	0,68223

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 71, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam.

Tabla 0-45 Resultados Red neuronal Multi-layer en función del número de épocas para el Problema de clasificación 1

Nº de épocas	Accuracy
10	0,67657
50	0,60415

100	0,68097
500	0,69108

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 71, un número de épocas de 100, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam variando su tasa de aprendizaje.

Tabla 0-46 Resultados Red neuronal Multi-layer en función de la tasa de aprendizaje para Adam para el Problema de clasificación 1

Tasa de aprendizaje	Accuracy
0,1	0,38796
0,01	0,46389
0,001	0,67545
0,0001	0,68852
0,00001	0,67167
0,000001	0,67436

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 71, un número de épocas de 100, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con tasa de aprendizaje 0,0001.

Tabla 0-47 Resultados Red neuronal Multi-layer en función del tamaño de lote para el Problema de clasificación 1

Tamaño de lote	Accuracy
10	0,62439
32	0,68517
62	0,68001
132	0,68313

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de neuronas en la capa intermedia de 71, un tamaño de lote de 32, un número de épocas de 100, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con tasa de aprendizaje 0,0001.

Tabla 0-48 Resultados Red neuronal Multi-layer en función del porcentaje para test para el Problema de clasificación 1

Porcentaje de test	Accuracy
0,1	0,69551
0,15	0,69427

0,2	0,69067
0,25	0,68548
0,3	0,68960

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de neuronas en la capa intermedia de 71, un tamaño de lote de 32, un número de épocas de 100, un porcentaje de test de 0,1, la función de activación ReLU en la capa intermedia y utilizando el método de optimización RMSprop variando la tasa de aprendizaje y momentum.

Tabla 0-49 Resultados Red neuronal Multi-layer en función de la tasa de aprendizaje y el momentum utilizando RMSprop para el Problema de clasificación 1

Tasa de aprendizaje	Momentum	Accuracy
0,1	0	0,38486
0,1	0,5	0,46329
0,1	0,9	0,46403
0,1	0,99	0,46534
0,01	0	0,46351
0,01	0,5	0,46247
0,01	0,9	0,46107
0,01	0,99	0,46360
0,001	0	0,63888
0,001	0,5	0,46209
0,001	0,9	0,46306
0,001	0,99	0,46183
0,00001	0	0,61701
0,00001	0,5	0,58321
0,00001	0,9	0,62042
0,00001	0,99	0,66121

En segundo lugar mostramos los resultados obtenidos para el *Problema de clasificación 2*.

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam.

Tabla 0-50 Resultados Red neuronal Multi-layer en función del número de neuronas para el Problema de clasificación 2

Nº de neuronas	Accuracy
2	0,52463
7	0,52463
12	0,52463
17	0,52463

22	0,52463
27	0,52463
32	0,52463
37	0,52463
42	0,71225
47	0,52463
52	0,71872
57	0,52463
62	0,52463
67	0,52463
72	0,71197
77	0,52463
82	0,70682
87	0,52463
92	0,71618

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 92, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam variando su tasa de aprendizaje.

Tabla 0-51 Resultados Red neuronal Multi-layer en función de la tasa de aprendizaje utilizando Adam para el Problema de clasificación 2

Tasa de aprendizaje	Accuracy
0,1	0,52476
0,01	0,52476
0,001	0,52476
0,0001	0,72070
0,00001	0,69726
0,000001	0,71675

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 92, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con una tasa de aprendizaje de 0,0001.

Tabla 0-52 Resultados Red neuronal Multi-layer en función del número de épocas para el Problema de clasificación 2

Nº de épocas	Accuracy
10	0,70171
50	0,72791
100	0,73459
500	0,73962

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 92, un número de épocas de 100, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con tasa de aprendizaje 0,0001.

Tabla 0-53 Resultados Red neuronal Multi-layer en función del tamaño de lote para el Problema de clasificación 2

Tamaño de lote	Accuracy
10	0,72340
32	0,72472
62	0,68815
132	0,71763

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de neuronas en la capa intermedia de 92, un tamaño de lote de 32, un número de épocas de 100, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con tasa de aprendizaje 0,0001.

Tabla 0-54 Resultados Red neuronal Multi-layer en función del porcentaje para test para el Problema de clasificación 2

Porcentaje de test	Accuracy
0,1	0,72640
0,15	0,73227
0,2	0,73526
0,25	0,72702
0,3	0,72479

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de neuronas en la capa intermedia de 92, un tamaño de lote de 32, un número de épocas de 100, un porcentaje de test de 0,2, la función de activación ReLU en la capa intermedia y utilizando el método de optimización RMSprop variando la tasa de aprendizaje y momentum.

Tabla 0-55 Resultados Red neuronal Multi-layer en función de la tasa de aprendizaje y el momentum para el Problema de clasificación 2

Tasa de aprendizaje	Momentum	Accuracy
0,1	0	0,52441
0,1	0,5	0,52318
0,1	0,9	0,47538
0,1	0,99	0,52516
0,01	0	0,52531
0,01	0,5	0,52338

0,01	0,9	0,47802
0,01	0,99	0,47576
0,001	0	0,52427
0,001	0,5	0,52346
0,001	0,9	0,52543
0,001	0,99	0,47577
0,00001	0	0,60897
0,00001	0,5	0,62830
0,00001	0,9	0,71382
0,00001	0,99	0,70048

En tercer lugar mostramos los resultados obtenidos para el *Problema de clasificación 3*.

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam.

Tabla 0-56 Resultados Red neuronal Multi-layer en función del número de neuronas para el Problema de clasificación 3

Nº de neuronas	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
2	0,00000	0	0,93313
7	0,00000	0	0,93329
12	0,00000	0	0,93554
17	0,00000	0	0,93364
22	0,00000	0	0,93375
27	0,00000	0	0,93309
32	0,00000	0	0,93363
37	0,00000	0	0,93354
42	0,00000	0	0,93285
47	0,00000	0	0,93321
52	0,00000	0	0,93302
57	0,00000	0	0,93440
62	0,00000	0	0,93136
67	0,00000	0	0,93405
72	0,00000	0	0,93193
77	0,00000	0	0,93430
82	0,00000	0	0,93158
87	0,00000	0	0,93456
92	0,00000	0	0,93055

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 47, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam.

Tabla 0-57 Resultados Red neuronal Multi-layer en función del número de épocas para el Problema de clasificación 3

Nº de épocas	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
10	0,00000	0	0,93049
50	0,00000	0	0,93285
100	0,00000	0	0,93080
500	0,00000	0	0,93222

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un tamaño de lote de 32, un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 47, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam variando su tasa de aprendizaje.

Tabla 0-58 Resultados Red neuronal Multi-layer en función de la tasa de aprendizaje para Adam para el Problema de clasificación 3

Tasa de aprendizaje	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
0,1	0,00000	0	0,91311
0,01	0,00000	0	0,91311
0,001	0,00000	0	0,93476
0,0001	0,00000	0	0,93391
0,00001	0,00000	0	0,87174
0,000001	0,00000	0	0,93179

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un porcentaje de test de 0,2, un número de neuronas en la capa intermedia de 47, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con tasa de aprendizaje 0,0001.

Tabla 0-59 Resultados Red neuronal Multi-layer en función del tamaño de lote para el Problema de clasificación 3

Tamaño de lote	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
10	0,00000	0	0,93286
32	0,00000	0	0,93045
62	1,00000	1	0,92246
132	0,00000	0	0,93322

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de neuronas en la capa intermedia de 47, un tamaño de lote de 62, un número de épocas de 50, la función de activación ReLU en la capa intermedia y utilizando el método de optimización Adam con tasa de aprendizaje 0,0001.

Tabla 0-60 Resultados Red neuronal Multi-layer en función del porcentaje para test para el Problema de clasificación 3

Porcentaje de test	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
0,1	0,00000	0	0,93238
0,15	0,00000	0	0,93343
0,2	0,00000	0	0,93161
0,25	0,00000	0	0,91232
0,3	0,00000	0	0,92068

Los resultados de la tabla que se muestra a continuación fueron obtenidos para un número de neuronas en la capa intermedia de 71, un tamaño de lote de 32, un número de épocas de 100, un porcentaje de test de 0,1, la función de activación ReLU en la capa intermedia y utilizando el método de optimización RMSprop variando la tasa de aprendizaje y momentum.

Tabla 0-61 Resultados Red neuronal Multi-layer en función de la tasa de aprendizaje y el momentum utilizando RMSprop para el Problema de clasificación 3

Tasa de aprendizaje	Momentum	Precision	Nº de muestras clasificadas como Regionalistas	Accuracy
0,1	0	0,00000	0	0,91258
0,1	0,5	0,00000	0	0,91254
0,1	0,9	0,00000	0	0,91259

0,1	0,99	0,00000	0	0,91273
0,01	0	0,00000	0	0,91397
0,01	0,5	0,00000	0	0,91419
0,01	0,9	0,00000	0	0,91199
0,01	0,99	0,00000	0	0,91268
0,001	0	0,00000	0	0,92948
0,001	0,5	0,00000	0	0,91261
0,001	0,9	0,00000	0	0,91424
0,001	0,99	0,00000	0	0,91330
0,00001	0	0,00000	0	0,92922
0,00001	0,5	0,00000	0	0,92942
0,00001	0,9	0,00000	0	0,91832
0,00001	0,99	0,00000	0	0,90316

L Código Python

En este anexo adjuntamos y explicamos el código Python utilizado para ejecutar los algoritmos. Todo el código que se muestra en este anexo ha sido introducido siguiendo las normas de estilo explicadas en el anexo Normas de estilo utilizadas para la introducción de código en la memoria.

Regresión lineal

Mostramos el programa básico que obtiene las métricas y el gráfico de una regresión lineal, tanto mediante línea como mediante parábola. El funcionamiento del programa queda detallado en los comentarios del propio código. Los programas de esta sección han sido ejecutados en local en el sistema operativo Ubuntu 20.

Regresión lineal mediante línea.

```
# Imports necesarios
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

#Configuración de la figura que contendrá la gráfica
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

#Carga los datos de entrada para la regresión
data = pd.read_csv("brausaeps_izq.csv", sep='\t')
dataX = data[['idh']]
X_train = np.array(dataX)
y_train = data['izq'].values

# Creamos el objeto de Regresión Lineal
regr = linear_model.LinearRegression()
```

```

#Lectura de los pesos de cada muestra en la regresión (obtenidos
previamente desde la base de datos)
pesos_archivo = pd.read_csv("pesos_todos_idh.csv",sep='\t')
pesos = pesos_archivo['PESOS'].values

# Entrenamos nuestro modelo
regr.fit(X_train, y_train, pesos)

# Realización de las predicciones
y_pred = regr.predict(X_train)

# Impresión por pantalla de los coeficientes y las métricas obtenidas
print('Coefficients: \n', regr.coef_)
print('Independent term: \n', regr.intercept_) # Este es el valor donde
corta el eje Y (en X=0)
print('MEAN SQUARE ERROR = ' + str(mean_squared_error(y_train, y_pred)))
# Error Cuadrado Medio
print('Variance score: ' + str(r2_score(y_train, y_pred))) #Varianza
print('NRMSE = ' +
str(((mean_squared_error(y_train,y_pred))**(1/2))/(y_train.max() -
y_train.min())) #NRMSE

#Realización del gráfico
f1 = data['idh'].values
f2 = data['izq'].values
fig = plt.figure()
pesos_dibujar = [(i*200)/(max(pesos)) for i in pesos]
plt.scatter(f1, f2,s=pesos_dibujar,c='blue')
plt.plot(f1, regr.coef_*f1 + regr.intercept_)
fig.suptitle('Reg. Lineal Izq', fontsize=20)
plt.xlabel('IDH', fontsize=18)
plt.ylabel('% Votos Izq', fontsize=16)
#plt.show()
plt.savefig('rgl_w_todos_izq.png') #Guardar la figura

```

Regresión lineal mediante parábola.

```

# Imports necesarios
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures

#Configuración de la figura que contendrá la gráfica
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

#Carga los datos de entrada para la regresión
data = pd.read_csv("brausaeps_der.csv",sep='\t')
dataX =data[['idh']]
X_train = np.array(dataX)
y_train = data['der'].values

# Creamos el objeto PolynomialFeatures necesario para utilizar una
parábola en la regresión
polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(X_train)

```

```

# Entrenamos nuestro modelo
pesos_archivo = pd.read_csv("pesos_todos_idh.csv", sep='\t')
pesos = pesos_archivo['PESOS'].values

# Creamos el objeto de Regresión Lineal
regr = linear_model.LinearRegression()

# Entrenamos nuestro modelo
regr.fit(x_poly, y_train, pesos)

# Realización de las predicciones
y_pred = regr.predict(x_poly)

# Impresión por pantalla de los coeficientes y las métricas obtenidas
print('Coefficients: \n', regr.coef_)
print('Independent term: \n', regr.intercept_)# Este es el valor donde
corta el eje Y (en X=0)
print('MEAN SQUARE ERROR = ' + str(mean_squared_error(y_train, y_pred)))
# Error Cuadrado Medio
print('Variance score: ' + str(r2_score(y_train, y_pred))) #Varianza
print('NRMSE = ' +
str(((mean_squared_error(y_train,y_pred))**(1/2))/(y_train.max() -
y_train.min())) #NRMSE

#Realización del gráfico
f1 = data['idh'].values
f2 = data['der'].values
fig = plt.figure()
pesos_dibujar = [(i*200)/(max(pesos)) for i in pesos]
plt.scatter(f1, f2,s=pesos_dibujar,c='blue')
plt.plot(f1, y_pred)
fig.suptitle('Reg. Lineal Der', fontsize=20)
plt.xlabel('IDH', fontsize=18)
plt.ylabel('% Votos Der', fontsize=16)
#plt.show()
plt.savefig('qrgl_w_todos_der.png') #Guardar la figura

```

Los diferentes problemas de regresión considerados se ejecutan sobre este mismo código, variando las líneas donde se leen los archivos de los datos y pesos,

```

data = pd.read_csv("brausaeps_der.csv", sep='\t')
pesos_archivo = pd.read_csv("pesos_todos_idh.csv", sep='\t')

```

Además, en cada ejecución se cambian los valores del título de la gráfica y de los ejes, dependiendo el problema de regresión y de la clase sobre la que se esté realizando.

Algoritmo KNN

Mostramos el programa básico que ejecuta el algoritmo KNN sin variación de parámetros dividido en sus correspondientes celdas (ya que ha sido escrito en Google Colab). El funcionamiento del programa queda detallado en las celdas de texto que encabezan cada celda de código y en los propios comentarios del código.

Imports

```

import sys
import numpy as np

```

```

import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from google.colab import drive
from sklearn.metrics import accuracy_score
from datetime import datetime
from __future__ import absolute_import, division, print_function,
unicode_literals

```

Definición de la función de carga de datos

```

#Función para cargar los datos normalizados y procesados mediante hot
encoding en arrays numpy divididos en conjunto de test y conjunto de
entrenamiento
#Devuelve las listas numpy: (x_columns,y_columns), (x_train, y_train),
(x_test, y_test)
#porcentaje_test: indica el porcentaje de los datos de muestra que se
utilizará para el conjunto de test
#data: DataFrame de la librería pandas con el dataset
#categoría: identifica la columna que es la categoría
#hot_encoded: lista con los nombres de las columnas que van a ser
procesadas por hot encoding
#normalizados: lista con los nombres de las columnas que van a ser
normalizadas
#seed: semilla para los procesos aleatorios que ocurren dentro de la
función
def
carga_datos_csv(data,categoría,porcentaje_test=0.25,hot_encoded=None,norm
alizados=None,seed=None):
    #Normalización
    for col in normalizados:
        data[col]=(data[col]-data[col].min())/(data[col].max()-
data[col].min())
    #Divide entre test y train
    train, test = train_test_split(data,
test_size=porcentaje_test,random_state=seed)
    #Separar el label
    label_train = pd.DataFrame(train[categoría])
    train.drop([categoría],axis=1, inplace=True)
    label_test = pd.DataFrame(test[categoría])
    test.drop([categoría],axis=1, inplace=True)
    #Hot encoding
    for col in hot_encoded:
        if col==categoría:
            label_train =
pd.concat([label_train,pd.get_dummies(label_train[col],
prefix=col)],axis=1)
            label_train.drop([col],axis=1, inplace=True)
            label_test =
pd.concat([label_test,pd.get_dummies(label_test[col],
prefix=col)],axis=1)
            label_test.drop([col],axis=1, inplace=True)
        else:
            train = pd.concat([train,pd.get_dummies(train[col],
prefix=col)],axis=1)
            train.drop([col],axis=1, inplace=True)
            test = pd.concat([test,pd.get_dummies(test[col],
prefix=col)],axis=1)
            test.drop([col],axis=1, inplace=True)

```



```

    #Rellenar el conjunto de entrenamiento con las columnas que estan en
    el conjunto de test pero no en el de entrenamiento y viceversa
    (características)
    missing_cols = set( train.columns ) - set( test.columns )
    for c in missing_cols:
        test[c] = 0
    test = test[train.columns]
    missing_cols = set( test.columns ) - set( train.columns )
    for c in missing_cols:
        train[c] = 0
    train = train[test.columns]
    #Rellenar el conjunto de test con las columnas que estan en el
    conjunto de entrenamiento pero no en el de test y viceversa (caterogías)
    missing_cols = set( label_train.columns ) - set( label_test.columns )
    for c in missing_cols:
        label_test[c] = 0
    label_test = label_test[label_train.columns]
    missing_cols = set( label_test.columns ) - set( label_train.columns )
    for c in missing_cols:
        label_train[c] = 0
    label_train = label_train[label_test.columns]

    #Obtener las listas ordenadas de las columnas de las características
    y las categorías
    x_columns= train.columns
    y_columns= label_train.columns

    #Convertir a array de numpy
    x_train = train.to_numpy()
    y_train = label_train.to_numpy()
    x_test = test.to_numpy()
    y_test = label_test.to_numpy()

    return (x_columns,y_columns), (x_train, y_train), (x_test, y_test)

```

Parámetros e hiperparámetros

```

test_ratio=0.2 #Porcentaje del total de los datos que se usará como
conjunto de test
dataset = '/content/drive/My
Drive/Archivos_google_colab/Datos/dataset_v1.csv' #Ruta al fichero csv
donde se encuentran los datos
n_neighbors = 7 #Numero de vecinos a considerar en KNN
metric = 'euclidean' #Distancia que usar en KNN

```

Carga de datos

```

#Montar el disco de google drive para poder acceder a los archivos
drive.flush_and_unmount()
drive.mount('/content/drive', force_remount=True)

#Llamada a la función de carga de datos
data = pd.read_csv(dataset, sep='\t')
label = 'CANDIDATURA'
hot =
['COMUNIDAD', 'PROVINCIA', 'GOBIERNO_COMUNIDAD', 'GOBIERNO_ESPANNA', 'GOBIERN
O_USA']
norm =
['RELIGIOSIDAD', 'EDAD_MEDIA', 'ESCOLARIZACION', 'ESPERANZA_VIDA', 'HABITANTE
S_MUNICIPIO', 'IDH',

```

```
'INTERNET','PIBPERCAPITA','CRIMINALIDAD','INMIGRACION_SUB','INMIGRACION_EUR','VIAJEROS','MUJERES','LATITUD','ALTITUD','LONGITUD']
(x_columns,y_columns), (x_train, y_train), (x_test, y_test) =
carga_datos_csv(data,label,test_ratio,hot,norm,seed=int(datetime.utcnow().timestamp()))
```

Creación y ejecución del modelo

```
#Creamos el modelo
knn = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
#Fase de entrenamiento
knn.fit(x_train, y_train)

#Fase de prueba
y_pred = knn.predict(x_test)

np.set_printoptions(threshold=sys.maxsize) #Mostrar todo el contenido de los arrays numpy

#Mostramos por pantalla la matriz de confusión y la accuracy obtenida
print(confusion_matrix(y_test, y_pred))
print(str(accuracy_score(y_test, y_pred)))
```

La variación de parámetros se realiza del siguiente modo. En la celda *Parámetros e hiperparámetros* hacemos que alguno de los parámetros (o varios) sean una lista de valores, por ejemplo

```
n_neighbors = [3,5,7,9]
```

Tras esto se altera la celda *Creación y ejecución del modelo* para que todo el bloque del código se ejecute dentro de un bucle que recorre los distintos valores del parámetro configurado como una lista. Se muestra por pantalla las métricas obtenidas y para qué valor de la lista se han obtenido estas. El problema de clasificación que se está considerando queda marcado por el archivo CSV que se lee en la línea

```
dataset = '/content/drive/My
Drive/Archivos_google_colab/Datos/dataset_v1.csv' #Ruta al fichero csv
donde se encuentran los datos
```

El archivo dataset_v1.csv contiene los datos del *Problema de clasificación 1*, el archivo dataset_v2.csv contiene los datos del *Problema de clasificación 2* y el archivo dataset_v3.csv contiene los datos del *Problema de clasificación 3*. Se ha realizado un programa distinto para cada problema de clasificación y para cada parámetro variado. Esto es así para poder ejecutar varios programas en paralelo.

Red neuronal Single-layer

Mostramos el programa básico que ejecuta la Red neuronal Single-Layer sin variación de parámetros dividido en sus correspondientes celdas (ya que ha sido escrito en Google Colab). El funcionamiento del programa queda detallado en las celdas de texto que encabezan cada celda de código y en los propios comentarios del código.

Imports

```

import numpy as np
from __future__ import absolute_import, division, print_function,
unicode_literals
try:
    %tensorflow_version 2.x
except Exception:
    pass
import sys
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import pandas as pd
from sklearn.model_selection import train_test_split
from google.colab import drive
from tensorflow import keras
from datetime import datetime
#Set de la semilla random global
tf.random.set_seed(1)

```

Definición de la función de carga de datos

```

#Función para cargar los datos normalizados y procesados mediante hot
encoding en arrays numpy divididos en conjunto de test y conjunto de
entrenamiento
#Devuelve las listas numpy: (x_columns,y_columns), (x_train, y_train),
(x_test, y_test)
#porcentaje_test: indica el porcentaje de los datos de muestra que se
utilizará para el conjunto de test
#data: DataFrame de la librería pandas con el dataset
#categoría: identifica la columna que es la categoría
#hot_encoded: lista con los nombres de las columnas que van a ser
procesadas por hot encoding
#normalizados: lista con los nombres de las columnas que van a ser
normalizadas
#seed: semilla para los procesos aleatorios que ocurren dentro de la
función
def
carga_datos_csv(data,categoría,porcentaje_test=0.25,hot_encoded=None,norm
alizados=None,seed=None):
    #Normalización
    for col in normalizados:
        data[col]=(data[col]-data[col].min())/(data[col].max()-
data[col].min())
    #Divide entre test y train
    train, test = train_test_split(data,
test_size=porcentaje_test,random_state=seed)
    #Separar el label
    label_train = pd.DataFrame(train[categoría])
    train.drop([categoría],axis=1, inplace=True)
    label_test = pd.DataFrame(test[categoría])
    test.drop([categoría],axis=1, inplace=True)
    #Hot encoding
    for col in hot_encoded:
        if col==categoría:
            label_train =
pd.concat([label_train,pd.get_dummies(label_train[col],
prefix=col)],axis=1)
            label_train.drop([col],axis=1, inplace=True)

```

```

        label_test =
pd.concat([label_test,pd.get_dummies(label_test[col],
prefix=col)],axis=1)
        label_test.drop([col],axis=1, inplace=True)
    else:
        train = pd.concat([train,pd.get_dummies(train[col],
prefix=col)],axis=1)
        train.drop([col],axis=1, inplace=True)
        test = pd.concat([test,pd.get_dummies(test[col],
prefix=col)],axis=1)
        test.drop([col],axis=1, inplace=True)
    #Rellenar el conjunto de entrenamiento con las columnas que estan en
el conjunto de test pero no en el de entrenamiento y viceversa
(características)
    missing_cols = set( train.columns ) - set( test.columns )
    for c in missing_cols:
        test[c] = 0
    test = test[train.columns]
    missing_cols = set( test.columns ) - set( train.columns )
    for c in missing_cols:
        train[c] = 0
    train = train[test.columns]
    #Rellenar el conjunto de test con las columnas que estan en el
conjunto de entrenamiento pero no en el de test y viceversa (categorías)
    missing_cols = set( label_train.columns ) - set( label_test.columns )
    for c in missing_cols:
        label_test[c] = 0
    label_test = label_test[label_train.columns]
    missing_cols = set( label_test.columns ) - set( label_train.columns )
    for c in missing_cols:
        label_train[c] = 0
    label_train = label_train[label_test.columns]

    #Obtener las listas ordenadas de las columnas de las características
y las categorías
    x_columns= train.columns
    y_columns= label_train.columns

    #Convertir a array de numpy
    x_train = train.to_numpy()
    y_train = label_train.to_numpy()
    x_test = test.to_numpy()
    y_test = label_test.to_numpy()

    return (x_columns,y_columns), (x_train, y_train), (x_test, y_test)

```

Parámetros e hiperparámetros

```

dataset='/content/drive/My
Drive/Archivos_google_colab/Datos/dataset_v1.csv' #Ruta al fichero csv
donde se encuentran los datos
epochs=10 #Numero de épocas (veces que cada lote pasa por la red neuronal
en la fase de entrenamiento)
lr=0.1 #Tasa de aprendizaje
momentum=0 #Momentum
test_ratio=0.30 #Porcentaje del total de los datos que se usará como
conjunto de test
metricas=['accuracy'] #Lo que queremos medir
inicializacion_pesos='glorot_normal' #Inicializacion del valor de los
pesos de las conexiones neuronales

```

```

batch=32 #Tamaño de lote (número de muestras que la red neuronal procesa
antes de cada actualización del gradiente)
loss='categorical_crossentropy' #Función de pérdida
#Metodo de optimización
#optimizador= keras.optimizers.SGD(learning_rate=lr,momentum=momentum)
optimizador= 'adam'

```

Carga de datos

```

#Montar el disco de google drive para poder acceder a los archivos
drive.flush_and_unmount()
drive.mount('/content/drive', force_remount=True)

#Llamada a la función de carga de datos
data = pd.read_csv(dataset, sep='\t')
label = 'CANDIDATURA'
hot =
['COMUNIDAD','PROVINCIA','GOBIERNO_COMUNIDAD','GOBIERNO_ESPANNA','GOBIERN
O_USA','CANDIDATURA']
norm =
['RELIGIOSIDAD','EDAD_MEDIA','ESCOLARIZACION','ESPERANZA_VIDA','HABITANTE
S_MUNICIPIO','IDH',
'INTERNET','PIBPERCAPITA','CRIMINALIDAD','INMIGRACION_SUB','INMIGRACION_E
UR','VIAJEROS','MUJERES']
(x_columns,y_columns), (x_train, y_train), (x_test, y_test) =
carga_datos_csv(data,label,test_ratio,hot,norm)

```

Creación y ejecución del modelo

```

#Creacion de la red neuronal
model = Sequential([
    Dense(len(x_train[0]),input_shape=(len(x_train[0]),)), # dense
layer de entrada
    Dense(len(y_train[0]),
activation='softmax',kernel_initializer=inicializacion_pesos), # dense
layer 2
])

model.compile(optimizer=optimizador, #El optimizador, es decir, el metodo
que se va a utilizar para que haya menos error
    #Adam es un tipo de gradient descent
    loss=loss, #Nombre de la funcion de coste
    metrics=metricas) #Métricas que quieres que el modelo
evaluate, le pedimos que evalúe la precision

#Entrenamiento
model.fit(x_train, y_train, epochs=epochs, batch_size=batch)

#Prediccion
results = model.evaluate(x_test, y_test, verbose = 0)

#Imprime las métricas obtenidas
print('test loss, test acc:', results)

```

La variación de parámetros se realiza del siguiente modo. En la celda *Parámetros e hiperparámetros* hacemos que alguno de los parámetros (o varios) sean una lista de valores, por ejemplo

```
epochs=[10,50,100,500]
```

Tras esto se altera la celda *Creación y ejecución del modelo* para que todo el bloque del código se ejecute dentro de un bucle que recorre los distintos valores del parámetro configurado como una lista. Se muestra por pantalla las métricas obtenidas y para qué valor de la lista se han obtenido estas. El problema de clasificación que se está considerando queda marcado por el archivo CSV que se lee en la línea

```
dataset = '/content/drive/My  
Drive/Archivos_google_colab/Datos/dataset_v1.csv' #Ruta al fichero csv  
donde se encuentran los datos
```

El archivo dataset_v1.csv contiene los datos del *Problema de clasificación 1*, el archivo dataset_v2.csv contiene los datos del *Problema de clasificación 2* y el archivo dataset_v3.csv contiene los datos del *Problema de clasificación 3*. Se ha realizado un programa distinto para cada problema de clasificación y para cada parámetro variado. Esto es así para poder ejecutar varios programas en paralelo.

En el caso especial del *Problema de clasificación 3* hemos escrito una función para obtener la matriz de confusión y de esta obtener la precisión. La función es la siguiente

```
def getCustomConfusionMatrix(y_test,y_pred):  
    confusion = [[0, 0],[0, 0]]  
    for i in range(len(y_pred)):  
        if y_pred[i][0] > y_pred[i][1]: #Ha predicho clase 0  
            if y_test[i][0]==1: #Predicho bien clase 0  
                confusion[0][0]+=1  
            else: #Predicho mal clase 0  
                confusion[0][1]+=1  
        else: #Ha predicho clase 1  
            if y_test[i][1]==1: #Predicho bien clase 1  
                confusion[1][1]+=1  
            else: #Predicho mal clase 1  
                confusion[1][0]+=1  
    return confusión
```

En los programas que realizan la variación de parámetros para este problema de clasificación, las categorías predichas por la red neuronal se obtienen mediante la línea de código

```
prediccion = model.predict(x_test)
```

Esta predicción se le pasa como argumento a nuestra función personalizada. Tras esto se imprimen la matriz de confusión y la precisión. Adjuntamos las líneas de código donde se realizan estos pasos

```
#Obtención de la matriz de confusión mediante nuestra función  
c_matrix = getCustomConfusionMatrix(y_test,prediccion)  
#Imprimir matriz de confusión y precision de la clase No-Regionalista  
print(str(c_matrix))  
print("Precision " + str(c_matrix[1][1]/(c_matrix[1][1] +  
c_matrix[1][0])) #precision
```

Red neuronal Multi-layer

Mostramos el programa básico que ejecuta la Red neuronal Multi-Layer sin variación de parámetros dividido en sus correspondientes celdas (ya que ha sido escrito en Google Colab). El funcionamiento del programa queda detallado en las celdas de texto que encabezan cada celda de código y en los propios comentarios del código.

Imports

```
import numpy as np
from __future__ import absolute_import, division, print_function,
unicode_literals
try:
    %tensorflow_version 2.x
except Exception:
    pass
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import pandas as pd
from sklearn.model_selection import train_test_split
from google.colab import drive
from tensorflow import keras
from datetime import datetime
#Set la semilla random global
tf.random.set_seed(1)
```

Definición de la función de carga de datos

```
#Función para cargar los datos normalizados y procesados mediante hot
encoding en arrays numpy divididos en conjunto de test y conjunto de
entrenamiento
#Devuelve las listas numpy: (x_columns,y_columns), (x_train, y_train),
(x_test, y_test)
#porcentaje_test: indica el porcentaje de los datos de muestra que se
utilizará para el conjunto de test
#data: DataFrame de la librería pandas con el dataset
#categoría: identifica la columna que es la categoría
#hot_encoded: lista con los nombres de las columnas que van a ser
procesadas por hot encoding
#normalizados: lista con los nombres de las columnas que van a ser
normalizadas
#seed: semilla para los procesos aleatorios que ocurren dentro de la
función
def
carga_datos_csv(data,categoria,porcentaje_test=0.25,hot_encoded=None,norm
alizados=None,seed=None):
    #Normalizacion
    for col in normalizados:
        data[col]=(data[col]-data[col].min())/(data[col].max()-
data[col].min())
    #Divide entre test y train
    train, test = train_test_split(data,
test_size=porcentaje_test,random_state=seed)
    #Separar el label
    label_train = pd.DataFrame(train[categoria])
    train.drop([categoria],axis=1, inplace=True)
    label_test = pd.DataFrame(test[categoria])
    test.drop([categoria],axis=1, inplace=True)
    #Hot encoding
    for col in hot_encoded:
```

```

        if col==categoria:
            label_train =
pd.concat([label_train,pd.get_dummies(label_train[col],
prefix=col)],axis=1)
            label_train.drop([col],axis=1, inplace=True)
            label_test =
pd.concat([label_test,pd.get_dummies(label_test[col],
prefix=col)],axis=1)
            label_test.drop([col],axis=1, inplace=True)
        else:
            train = pd.concat([train,pd.get_dummies(train[col],
prefix=col)],axis=1)
            train.drop([col],axis=1, inplace=True)
            test = pd.concat([test,pd.get_dummies(test[col],
prefix=col)],axis=1)
            test.drop([col],axis=1, inplace=True)
        #Rellenar el conjunto de entrenamiento con las columnas que estan en
el conjunto de test pero no en el de entrenamiento y viceversa
(características)
        missing_cols = set( train.columns ) - set( test.columns )
        for c in missing_cols:
            test[c] = 0
        test = test[train.columns]
        missing_cols = set( test.columns ) - set( train.columns )
        for c in missing_cols:
            train[c] = 0
        train = train[test.columns]
        #Rellenar el conjunto de test con las columnas que estan en el
conjunto de entrenamiento pero no en el de test y viceversa (caterogías)
        missing_cols = set( label_train.columns ) - set( label_test.columns )
        for c in missing_cols:
            label_test[c] = 0
        label_test = label_test[label_train.columns]
        missing_cols = set( label_test.columns ) - set( label_train.columns )
        for c in missing_cols:
            label_train[c] = 0
        label_train = label_train[label_test.columns]

        #Obtener las listas ordenadas de las columnas de las características
y las categorías
        x_columns= train.columns
        y_columns= label_train.columns

        #Convertir a array de numpy
        x_train = train.to_numpy()
        y_train = label_train.to_numpy()
        x_test = test.to_numpy()
        y_test = label_test.to_numpy()

        return (x_columns,y_columns), (x_train, y_train), (x_test, y_test)

```

Parámetros e hiperparámetros

```

dataset='/content/drive/My
Drive/Archivos_google_colab/Datos/dataset_v1.csv' #Ruta al fichero csv
donde se encuentran los datos
epochs=10 #Numero de épocas (veces que cada lote pasa por la red neuronal
en la fase de entrenamiento)
lr=0.1 #Tasa de aprendizaje
momentum=0 #Momentum

```



```

test_ratio=0.30 #Porcentaje del total de los datos que se usará como
conjunto de test
metricas=['accuracy'] #Lo que queremos medir
inicializacion_pesos=['glorot_normal','glorot_normal'] #Inicializacion
del valor de los pesos de las conexiones neuronales (una lista porque hay
varias capas)
batch=32 #Tamaño de lote (número de muestras que la red neuronal procesa
antes de cada actualización del gradiente)
activation_layer_1 = 'relu' #Funcion de activación de las neuronas de la
capa intermedia
nneurons_layer_1 = 32 #Número de neuronas de la capa intermedia
loss='categorical_crossentropy' #Función de pérdida
#Metodo de optimización
#optimizador= keras.optimizers.SGD(learning_rate=lr,momentum=momentum)
optimizador= 'adam'

```

Carga de datos

```

#Montar el disco de google drive para poder acceder a los archivos
drive.flush_and_unmount()
drive.mount('/content/drive', force_remount=True)

#Llamada a la función de carga de datos
data = pd.read_csv(dataset, sep='\t')
label = 'CANDIDATURA'
hot =
['COMUNIDAD', 'PROVINCIA', 'GOBIERNO_COMUNIDAD', 'GOBIERNO_ESPANNA', 'GOBIERN
O_USA', 'CANDIDATURA']
norm =
['RELIGIOSIDAD', 'EDAD_MEDIA', 'ESCOLARIZACION', 'ESPERANZA_VIDA', 'HABITANTE
S_MUNICIPIO', 'IDH',

'INTERNET', 'PIBPERCAPITA', 'CRIMINALIDAD', 'INMIGRACION_SUB', 'INMIGRACION_E
UR', 'VIAJEROS', 'MUJERES', 'LATITUD', 'ALTITUD', 'LONGITUD']
(x_columns, y_columns), (x_train, y_train), (x_test, y_test) =
carga_datos_csv(data, label, test_ratio, hot, norm)

```

Creación y ejecución del modelo

```

#Creacion de la red neuronal
model = Sequential([
    Dense(len(x_train[0]), input_shape=(len(x_train[0]),)), # dense layer
de entrada
    Dense(nneurons_layer_1,
activation=activation_layer_1, kernel_initializer=inicializacion_pesos[0])
, # dense layer 1
    Dense(len(y_train[0]),
activation='softmax', kernel_initializer=inicializacion_pesos[1]), #
dense layer 2
])

model.compile(optimizer=optimizador, #El optimizador, es decir, el metodo
que se va a utilizar para que haya menos error
                loss=loss, #Nombre de la funcion de coste
                metrics=metricas) #Métricas que quieres que el modelo
evaluate

#Entrenamiento
model.fit(x_train, y_train, epochs=epochs, batch_size=batch)

```

```
#Prediccion
results = model.evaluate(x_test, y_test, verbose = 0)

#Imprime las métricas obtenidas
print('test loss, test acc:', results)
```

La variación de parámetros se realiza del siguiente modo. En la celda *Parámetros e hiperparámetros* hacemos que alguno de los parámetros (o varios) sean una lista de valores, por ejemplo

```
epochs=[10,50,100,500]
```

Tras esto se altera la celda *Creación y ejecución del modelo* para que todo el bloque del código se ejecute dentro de un bucle que recorre los distintos valores del parámetro configurado como una lista. Se muestra por pantalla las métricas obtenidas y para qué valor de la lista se han obtenido estas. El problema de clasificación que se está considerando queda marcado por el archivo CSV que se lee en la línea

```
dataset = '/content/drive/My
Drive/Archivos_google_colab/Datos/dataset_v1.csv' #Ruta al fichero csv
donde se encuentran los datos
```

El archivo dataset_v1.csv contiene los datos del *Problema de clasificación 1*, el archivo dataset_v2.csv contiene los datos del *Problema de clasificación 2* y el archivo dataset_v3.csv contiene los datos del *Problema de clasificación 3*. Se ha realizado un programa distinto para cada problema de clasificación y para cada parámetro variado. Esto es así para poder ejecutar varios programas en paralelo.

En el caso especial del *Problema de clasificación 3* se utiliza la función para obtener la matriz de confusión y los procedimientos descritos en la sección anterior.

